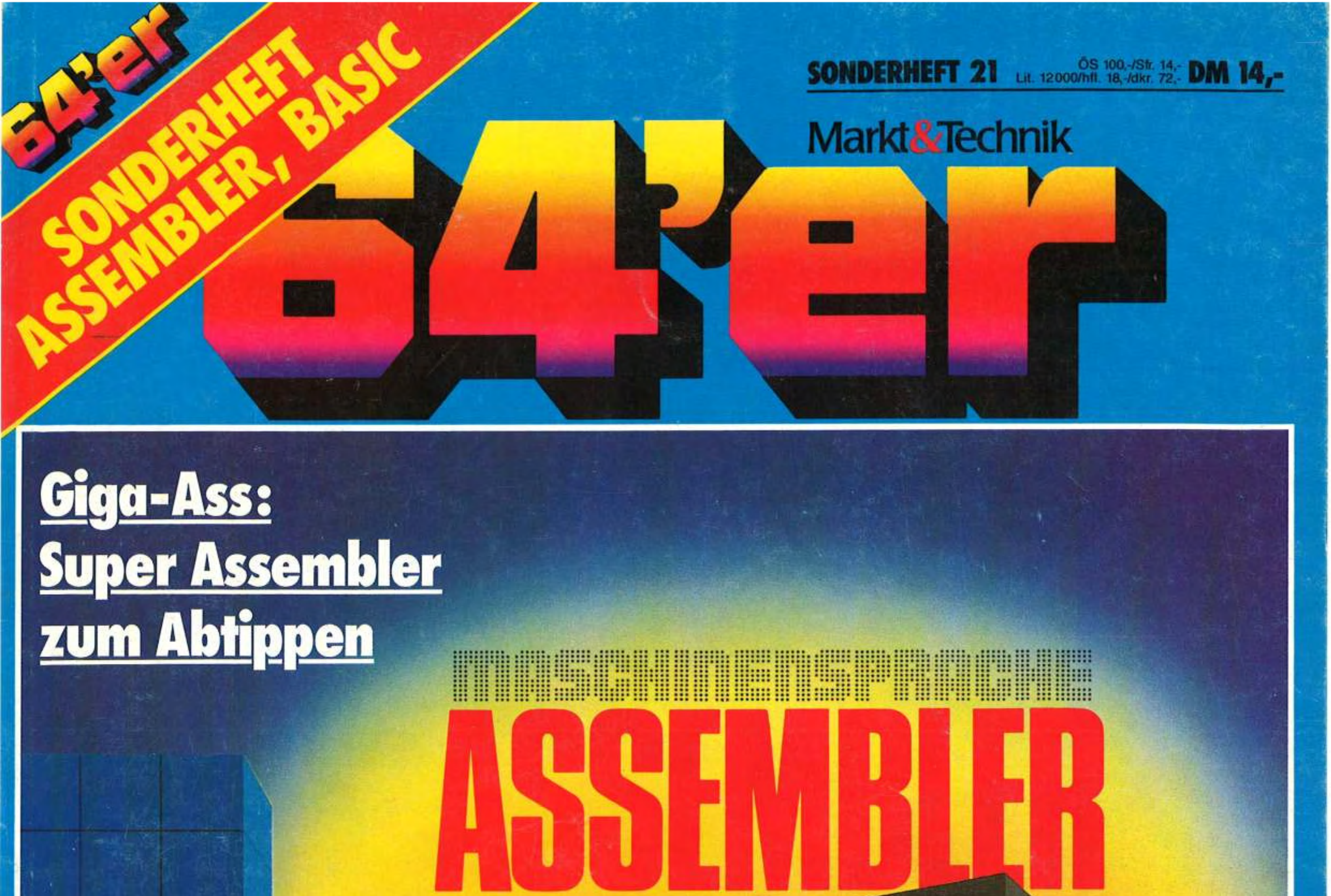


Giga-Ass: Super Assembler zum Abtippen

MASCHINENSPRACHE

ASSEMBLER

Top Listings

- ★ Makro-Bibliothek: nützliche Assembler-Routinen
- ★ Das Commodore-512-KByte-Modul jetzt auch am C64
- ★ Bewegte Farbgrafik im Bildschirmrahmen

Keine Angst vor Maschinensprache

Ausführliche Kurse zum Mitmachen:

- ★ So lernt jeder Assembler
- ★ Von Basic zu Assembler

Alle Programme auch auf Diskette erhältlich

PROTEXT

64ER ONLINE

Nur Fliegen ist schöner

Wenn Sie die Wahl hätten zwischen einem Auto und einem Sportflugzeug, um ein Ziel möglichst schnell zu erreichen, für welches Transportmittel würden Sie sich entscheiden? Wohl für das Flugzeug. Fliegen will aber gelernt sein! Daher scheidet für die meisten von uns diese Möglichkeit der Fortbewegung aus.

Ähnlich verhält es sich mit Basic und Assembler (Maschinensprache) auf dem C64: Die meisten Problemstellungen und Aufgaben lassen sich in Basic lösen. Nur ist das Resultat in der Geschwindigkeit und Flexibilität oft nicht befriedigend. Und manches geht eben gar nicht – ebensowenig wie Sie allein mit dem Auto den Ozean überqueren können!

In dieser Analogie ist auch der Grund zu suchen, warum Assembler für die einen die »einzig wahre Art« der Programmierung, für die anderen (noch) ein Buch mit sieben Siegeln darstellt: Die Kunst der Assembler-Programmierung ist nicht ganz einfach zu beherrschen und will erlernt sein.

Daher haben wir für Sie dieses Sonderheft zusammengestellt, das die Verbindung zwischen dem vertrauten Basic Ihres C64 und der Assembler-Sprache herstellt.

Für Einsteiger in die »hohe Schule« der Assembler-Programmierung bieten wir mit dem Kurs »Keine Angst vor Maschinensprache« einen Beitrag, der Schritt für Schritt in diese Materie einführt.

Haben Sie die ersten Hürden genommen, so steht Ihrem Wechsel ins Lager der Assembler-Alchimisten nichts mehr entgegen. In dem Kurs »Von Basic zu Assembler« greifen wir in unsere Trickkiste. Denn eins ist wichtig zu wissen: Durch die Kenntnis der einzelnen Assembler-Befehle allein kann man noch nicht programmieren. Dies gilt zwar für alle Computersprachen, aber in erhöhtem Maße für Assembler. Denn nur durch das Nachvollziehen und Analysieren anderer Programme, sowie durch Tips & Tricks eines erfahrenen Programmierers lernt man, das erworbene Wissen richtig anzuwenden. Genau das bewirkt dieser Kurs.

Neben den erwähnten Kursen hält dieses Sonderheft noch einige andere »Juwelen« für Sie bereit:

An erster Stelle ist hier der Makro-Assembler »Giga-Ass« zu nennen. Er tritt die Nachfolge unseres bewährten Assemblers Hypra-Ass an und läßt durch seinen Komfort das Programmieren in Assembler zum Vergnügen werden.

Ebenfalls hervorzuheben ist die Makro-Bibliothek für den Hypra-Ass. In Ihr finden Sie zahlreiche Routinen und Makros mit häufig benötigten Funktionen. Diese Bibliothek erspart Ihnen, bei Programmprojekten jeweils das Rad »neu erfinden« zu müssen.

Eine geradezu »klassische« Anwendung der Assembler-Sprache, um faszinierende Effekte im Bildschirmrahmen zu erzielen, ist der Rasterzeilen-Interrupt. Das Programm »Magic Border Beams« erlaubt es nun, mit einem komfortablen Editor farbige Effekte zu entwerfen und in eigene Programme einzubinden. Es lassen sich mit »Magic Border Beams« Filme mit bis zu 256 verschiedenen Bildsequenzen entwerfen, die sogar fließend wirkende Farbübergänge erlauben.

Auch an die Druckerfans wurde gedacht. Mit »Obsess V 3.1« lassen sich mit jedem Commodore- oder Epson-kompatiblen Drucker farbige Hardcopies von Multicolor-Malprogrammen wie Paint-Magic oder Koala-Painter ausdrucken. Auch hier befindet sich, wie bei nahezu allen Programmen, der dokumentierte Assembler-Quellcode auf der Programmservice-Diskette, so daß Sie demnächst Ihr Hardcopy-Wunschprogramm selbst schreiben können.

Last not least stellt Ihnen dieses Sonderheft drei Basic-Erweiterungen zur Verfügung, von denen jede ihre ganz speziellen Vorzüge hat:

So ist es mit »DMA-Basic« erstmals möglich, die Commodore-Speichererweiterungen für den C128 am C64 zu verwenden. Mit der Speichererweiterung 1750 stehen Ihnen damit sage und schreibe 576 KByte am C64 zur Verfügung – mehr als mancher Personal-Computer zu bieten hat!

»Paradoxon-Basic« bietet Ihnen 50 KByte freien Basic-Speicher und eine Reihe verbesserter Befehle. Berechnete Zeilennummern bei GOSUB und GOTO seien hier nur ein Beispiel.

»Rekursiv-Basic« ist wieder von anderer Qualität: Rekursive Programmierung, Definitionen von Prozeduren und neuen Befehlen werden damit zum Kinderspiel.

Ein umfangreicher und sehr nützlicher Tabellenteil mit einer Übersicht aller Assembler-Befehle der 6510-CPU und der wichtigsten ROM-Routinen runden dieses Sonderheft ab und machen es sowohl für den Einsteiger als auch für den Assembler-Profi zu einem unverzichtbaren Wegbegleiter beim Streifzug durch die faszinierende Welt der Assembler-Programmierung.

Ihr Klaus Schrödl, Redakteur

64'er SONDERHEFT

PROGRAMM-SERVICE

Direkt bestellen statt abtippen!

Die aktuelle Diskette zum Heft:

Assembler-Programmierung einfach wie in Basic

GIGA-ASS: Ein neu gestalteter Makro-Assembler der Spitzenklasse erlaubt es, Maschinensprache-Programme so komfortabel wie in Basic zu schreiben. Durch ein mitgeliefertes Konvertierungsprogramm können auch Hypra-Ass-Quelltexte mit Giga-Ass verarbeitet werden. Eine große Hilfe beim Schreiben eigener Maschinenprogramme ist auch die Makro-Bibliothek auf Diskette, die oft verwendete Routinen bereitstellt. Auch für den Basic-Programmierer wird einiges geboten: **DMA-BASIC.** Mit diesem Programm können Sie die 512-KByte-RAM-Erweiterung von Commodore am C 64 verwenden und somit über 576 KByte Speicherplatz verfügen. **PARADOXON-BASIC.** Eine weitere Basic-Erweiterung, stellt dem Basic Programmierer 50 KByte für seine Programme und eine Reihe neuer mächtiger Befehle zur Verfügung. **REKURSIV-BASIC.** Diese Erweiterung ist von einer anderen Qualität: Sie erlaubt, wie der Name schon sagt, rekursive Programmierung durch einen vergrößerten Stack und vor allem die Definition von Prozeduren wie in Pascal. **OBSESS V3.1.** Dieses Programm erlaubt es, auf einfache Weise mit normalen Matrixdruckern Mullticolor-Grafiken in strahlenden Farben auch zu Papier zu bringen. Ergänzt werden die Programme durch eine Reihe von Tips&Tricks. Für den ambitionierten Programmierer befinden sich des weiteren sämtliche Quell-Codes der Maschinenprogramme auf Diskette. Natürlich enthält die Programmservice-Diskette auch alle Programme, die mit einem Diskettensymbol gekennzeichnet sind.

1 Diskette für C 64/C 128
Best.-Nr. 15721 — sFr 24,90/öS 299,-* **DM 29,90***

Bestellnummern für Disketten zum 64'er-Magazin

Programmservice-Disketten sind zu allen Ausgaben des 64'er-Magazins erhältlich. Bitte geben Sie auf der in diesem Heft abgedruckten Zahlkarte die Bestellnummer an. Diese Nummer setzt sich wie folgt zusammen:

64'er-Sonderhefte ab Ausgabe 13/1987:

Konstant		Jahr	Ausgabe	
1	5	7	1	6

z.B.: 15716 für die Diskette zum Sonderheft 16/1987

64'er-Magazin, Ausgaben 1/85 bis 12/86:

Konstant		Jahr		Ausgabe		
L	6	8	6	0	6	D

z.B.: L6 86 06 D für die Diskette zur Ausgabe 6/1986.

64'er-Magazin ab Ausgabe 1/1987:

Konstant		Jahr	Ausgabe	
1	0	7	0	1

z.B.: 10701 für die Diskette zur Ausgabe 1/1987.

Der gute Geist für Ihre Floppy 1541

Disk-Demon. Disk-Demon ist ein Diskettenmonitor ganz besonderer Art. Wenn Sie Probleme mit fehlerhaften Disketten haben, auf denen wichtige Daten gespeichert sind, oder Sie eine Diskette lediglich einmal genauer unter die Lupe nehmen wollen, dann ist der Disk-Demon genau das richtige Werkzeug für Sie. Er liest, analysiert und repariert defekte Sektoren, bearbeitet die Spuren 0 bis 42 auf einer Diskette und unterstützt Sie auch dann, wenn es um die Entwicklung eigener Kopierschutzmethoden geht. **Hi-Eddi+ mit Maus.** Mit diesen Routinen kann man die Proportional-Maus von Reisware für die Steuerung von Hi-Eddi+ verwenden. **Hirn 64.** Hirn 64 ist ein mit Hypra-Basic geschriebenes Spiel, das auch Sie in seinen Bann ziehen wird. Es ist eine grafisch ausgefeilte Variante des bekannten Masterminds, auch als Superhirn bekannt. Die Beschreibung finden Sie in Ausgabe 8/87

Diskette für C 64/C 128, Best.-Nr. 10708 — sFr 24,90/öS 299,-* **DM 29,90***

Mastertext 128 – kaum zu übertreffen

Master-Text 128. Das super-professionelle Master-Text für den C 128 bietet professionelle Leistungsmerkmale. Durch Menü- und Window-Steuerung ist das Programm anwenderfreundlich und bietet neben dem Standard an Befehlen noch Textbaustein-Funktionen, einen Terminal-Modus, einen integrierten Taschenrechner sowie eine Uhr mit Alarmfunktion. **Textas.** Das Textverarbeitungsprogramm Textas läuft auf dem C 64 und ist speziell für den MPS 801 und kompatible Drucker entwickelt worden. Mit selbstdefinierbarem Zeichensatz stellt Textas nun auch die deutschen Sonderzeichen zur Verfügung, bietet eine deutlich bessere Druckqualität und erlaubt sogar das Einbinden von Hires-Grafiken und Sprites. **MacMatrix.** Mit MacMatrix stellen wir Ihnen ein Programm zur Verfügung, mit dem Sie auf komfortable Art und Weise NLQ-Zeichensätze für Ihren NL-10 mit Commodore-Interface entwerfen können. Außerdem sind noch viele Tips und Tricks, zum Beispiel für Vizawrite, auf der Diskette enthalten. Die Beschreibungen finden Sie im Sonderheft Ausgabe 18/87 (Drucker).

1 Diskette für C 64/C 128, Best.-Nr. 15718 — sFr 24,90/öS 299,-* **DM 29,90***

Weitere Angebote zum Thema Assembler

Kontrolle über jedes BIT im Speicher

Promon 64. Promon 64 ist ein vollkommen überarbeiteter SMON, der über einen riesigen und leistungsfähigen Befehlssatz verfügt. Neben den Befehlen des SMON können Sie Hires-Grafiken suchen und ASCII-Tabellen eingeben. Illegale Opcodes können sowohl assembliert als auch disassembliert werden. Ein Diskettenmonitor, auf dem sich alle Befehle von Promon 64 anwenden lassen, ist integriert. **CP/M <-> CBM.** Mit diesem Programm wird zum ersten Mal die Möglichkeit geboten, Daten und Programme auf einfachem Weg zwischen dem CP/M- und C 64-Format beliebig hin und her zu transferieren. Außerdem viele Tips und Tricks für den C 64. Die Beschreibungen finden Sie im Sonderheft Ausgabe 12/86 (Assembler, Programmiersprachen).

1 Diskette für den C 64, Best.-Nr. L686S12D — sFr 24,90/öS 299,-* **DM 29,90***

Ein Muß für Assembler-Programmierer

SMON. SMON gehört zu den besten erhältlichen Monitoren. Er zeichnet sich besonders durch seine Vielzahl von Befehlen, seine Arbeitsgeschwindigkeit und seinen Bedienungskomfort aus. **Hypra-Ass.** Hypra-Ass ist ein rein in Maschinensprache geschriebener Drei-Paß-Makroassembler mit integriertem Editor für C 64 mit Floppy 1541. Hypra-Ass gehört, was Leistung und Schnelligkeit betrifft, zu den besten Assemblern. **Re-Ass.** Mit diesem Programm können Sie Maschinenprogramme wieder in Quellcode für den Hypra-Ass zurückverwandeln. Außerdem viele Tips und Tricks für den C 64. Die Beschreibungen finden Sie im Sonderheft Ausgabe 8/85 (Assembler).

1 Diskette für C 64, Best.-Nr. L685S8D — sFr 24,90/öS 299,-* **DM 29,90***

Von Profi-Ass zu Hypra-Ass

Pth-Trans. Mit Pth-Trans können Sie Quelltexte, die mit dem Profi-Ass erzeugt wurden, einfach und problemlos ins Hypra-Ass-Format wandeln. **Kudi 64.** Dieses Programm kann mit einigen neuen Basic-Befehlen eine vollständige Kurvendiskussion durchführen. Es errechnet Ableitungen, Nullstellen, Extrema sowie Definitionslücken und stellt diese grafisch dar. **Trickfilm mit dem C 64.** Dieser Trickfilmgenerator konvertiert Hires-Grafiken in den Lores-Bildschirm und bietet Editor-Funktionen für Filmschnitte und zum Erstellen von Filmsequenzen. Auf der Programmservice-Diskette finden Sie zusätzlich zwei eindrucksvolle Filme. **Split-Screen.** Split-Screen ist eine Erweiterung zu Hypra-Basic, mit deren Hilfe man den Bildschirm an beliebiger Stelle zwischen Grafik- und Textbildschirm teilen kann. Zusätzlich viele Utilities für den C 64/C 128 sowie C 16. Die Beschreibungen finden Sie in Ausgabe 2/87.

Diskette für C 64, Best.-Nr. 10702 — sFr 24,90/öS 299,-* **DM 29,90***

Maschinensprache für Basic-Programmierer

Ascopiler 64. Ascopiler 64 ist ein kurzer und schnell arbeitender 3-Paß-Compiler, der ein vereinfachtes, sogenanntes Tiny-Basic in reinen Maschinencode übersetzt. Dadurch lassen sich zeitkritische Routinen in Basic-Programmen, ohne Kenntnis von Assembler, erheblich beschleunigen. **Datawork-Basic.** Diese Basic-Erweiterung erleichtert in erster Linie das Programmieren von Dateiprogrammen, besonders derjenigen, die mit Bildschirmmasken arbeiten. Hierzu stehen 22 neue Befehle zur Verfügung. **Life.** Life ist wohl eine der faszinierendsten Simulationen von biologischen Vorgängen auf Computern. Mit Life lassen sich evolutionäre Abläufe spielerisch verstehen lernen. **Border-Clock.** Dieses Programm stellt während des Editierens oder Ablaufs von Basic- oder Maschinenspracheprogrammen im unteren Rand des Bildschirms eine absolut genaue Uhr mit Stunden-, Minuten und Sekundenanzeige dar. Zusätzlich viele Tips und Tricks für den C 64. Die Beschreibungen finden Sie in Ausgabe 1/86.

Diskette für C 64, Best.-Nr. L68601D — sFr 24,90/öS 299,-* **DM 29,90***

Makro-Assembler der Spitzenklasse

Hypra-Ass. Hypra-Ass ist ein rein in Maschinensprache geschriebener Drei-Paß-Makroassembler mit integriertem Editor für C 64 mit Floppy 1541. Hypra-Ass gehört, was Leistung und Schnelligkeit betrifft, zu den besten Assemblern. **Super Term 64.** Dieses Terminalprogramm zeichnet sich besonders durch seine hohe Benutzerfreundlichkeit und einen großen Terminal-Speicher aus. **Modulator.** Mit diesem Programm können Sie nun sämtliche Parameter des SID übersichtlich einstellen und den SID als Synthesizer verwenden. Weiterhin viele Tips und Tricks für den C 64 und C 16. Die Beschreibungen finden Sie in Ausgabe 7/85.

Diskette für C 64, Best.-Nr. L68507A — sFr 24,90/öS 299,-* **DM 29,90***

*inkl. MwSt. Unverbindliche Preisempfehlung

Bestellungen bitte an: Markt&Technik Verlag AG, Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, D-8013 Haar, Telefon (089) 4613-0. **Schweiz:** Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 415656. **Österreich:** Ueberreuter Media Handels- und Verlagsgesellschaft mbH (Großhandel), Alser Straße 24, A-1091 Wien, Telefon (0222) 481538-0; Microcomputique E. Schiller, Fasangasse 24, A-1030 Wien, Telefon (0222) 785661; Bücherzentrum Meidling, Schönbrunner Straße 261, A-1120 Wien, Telefon (0222) 833196. **Bestellungen aus anderen Ländern bitte nur schriftlich an:** Markt&Technik Verlag AG, Abt. Buchvertrieb, Hans-Pinsel-Straße 2, D-8013 Haar, und gegen Bezahlung einer Rechnung im voraus.

Bitte verwenden Sie für Ihre Bestellung und Überweisung die abgedruckte Postgiro-Zahlkarte, oder senden Sie uns einen Verrechnungs-Scheck mit Ihrer Bestellung. Sie erleichtern uns die Auftragsabwicklung, und dafür berechnen wir Ihnen keine Versandkosten.

INHALT

Alle Programme aus Artikeln mit dem ■-Symbol finden Sie auf der Programmservice-Diskette zu diesem Sonderheft (siehe linke Seite)

Alles über den Commodore 64

Dieser erste Band aus der Commodore-Sachbuchreihe von Markt & Technik ist ein Standardwerk zum C64. Erstmals auf der Hannover Messe 1984 angeboten, erfüllt dieses nun überarbeitete Programmierhandbuch einen lange gehegten Wunsch vieler Programmierer: Der englischsprachige »Programmers Reverence Guide« wurde ins Deutsche übersetzt.

Das über 500 Seiten starke Programmierhandbuch ist in vier wesentliche Abschnitte gegliedert: die Programmierung in Basic, in Assembler, die Beschreibung der Hardware und das neue Betriebssystem »Geos«. In allen Abschnitten wird auf die besonderen Baugruppen des C64 für Grafik, Musik, Ein-/Ausgabe und deren Programmierung eingegangen. Dem Konzept folgend, kein Lehrbuch, sondern mehr ein Nachschlagewerk zu sein, sind die angeführten Beispiele relativ kurz gehalten.

Gerade in der Kürze der Darstellung liegt der wichtigste Vorteil dieses Buches: Es fällt einem sehr leicht, Informationen zu den verschiedensten Problemen zu finden.

Hinten im Buch ist sogar ein kompletter Schaltplan des C64 eingeheftet.

Es gibt kaum ein Buch zum C64, das gleichermaßen umfassend informiert und trotzdem leicht verständlich geschrieben ist. Das Programmierhandbuch »Alles über den Commodore 64« sollte neben jedem C64 liegen, auch wenn 59 Mark zusätzlich investiert werden müssen.

(aw/sk)

»Alles über den Commodore 64«, Markt & Technik Verlag, Commodore Sachbuchreihe Band 1, 513 Seiten, ISBN 3-89090-379-7, Preis 59 Mark

C64 – Programmieren in Maschinensprache

Dieses Buch spricht den Leserkreis an, der sich schon gut in Basic eingearbeitet hat und der jetzt mit der fortgeschrittenen Programmierung in Maschinensprache seine Programme optimieren will. Dazu bietet das Buch zahlreiche fertig ausgetestete Programme aus allen Anwendungsbereichen wie beispielsweise Grafik, Floppy und Datenverwaltung. Der Anwender muß die für ihn zutreffende Problemlösung nur noch von der mitgelieferten Diskette laden und in sein eigenes Programm einbinden.

Das Buch ist aber nicht nur als Softwarebibliothek zu betrachten, es enthält zudem zahlreiche Beispiele und Aufgaben, die den Sinn haben, das in den Programmen vermittelte Wissen weiter auszubauen und zu vertiefen. Ganz hervorragend werden die Ein- und Ausgaberoutinen und sämtliche Arithmetikfunktionen erklärt.

Weiterhin findet man im Anhang eine Tabelle mit dem vollständigen Befehlssatz des 6502-Prozessors; jeder Befehl ist mit einem Beispiel versehen, das die Wirkung auf die Flags gut darstellt. Genauso ausführlich werden die wichtigsten Einsprungadressen in das Betriebssystem beschrieben. Somit erhält sowohl der weniger Versierte als auch der Profi für einen Preis von 52 Mark mit dem Buch und der Diskette sowohl ein inhaltstarkes Nachschlagewerk, das seinesgleichen sucht als auch eine wahre Fundgrube an oft benötigten Assembler-Routinen.

(Udo Reetz/sk)

Winfried und Frank Kamera: C 64 – Programmieren in Maschinensprache, Markt & Technik Verlag, 327 Seiten, ISBN 3-89090-168-9, Preis 52 Mark

64 Intern

Obwohl Data Becker in den letzten Jahren den Computer-Literaturmarkt mit einer Vielzahl von Werken überschwemmt hat, gehört eines der ersten Bücher zu den besten: Es befreite den mit seinem Handbuch alleine gelassenen C64-Besitzer von seinen Informationsdefizit.

1986 erschien eine überarbeitete und erweiterte Auflage von »64 Intern«, die auch das neue Betriebssystem »Geos« berücksichtigt.

Ähnlich wie ein Systemhandbuch aufgebaut, soll »64 Intern« durch gute Beschreibung der Hardware und ihrer Programmierung ein ständiger Begleiter beim Programmieren sein. Diesen Anspruch unterstreicht das kommentierte ROM-Listing und der Schaltplan des C64. Die wichtigsten Kapitel dieses Buches sind:

- Hardware (CPU, Speicherbelegungspläne, User-Port, Expansion-Port)
- Tonprogrammierung (der SID, Register, A/D-Wandler, Synthy 64)
- Grafik (der VIC, Registerbeschreibung, Betriebsarten, Schnittstellen zum Prozessor)
- Ein-/Ausgabebausteine (Register-Plan, Ports, Timer, die CIAs, Joystickprogrammierung)
- Der Basic-Interpreter (Erweiterung des Basic, Monitor-Programm, wichtige Kernel-Adressen, RS232, serieller Bus)
- Vergleich: VC 20 - CBM - C64
- ROM-Listing

»64 Intern« kann mit Fug und Recht als eines der Standardwerke für den Assembler-Programmierer auf dem C64 bezeichnet werden. (aw/sk)

Brückmann, Englisch u.a.: 64 Intern, Data Becker, 1986, ISBN 3-89011-000-2, 628 Seiten, Preis 69 Mark

C64 Computer-Handbuch

Auf der Titelseite heißt es: »Einführung und Referenz für kompetentes Arbeiten«. Und das ist es in der Tat. Das Buch ist eine deutsche Übersetzung und Bearbeitung der englischen Originalausgabe »Programming the Commodore 64« und kann getrost als eines der wenigen Standardwerke zum C64 gelten.

Auf 600 Seiten findet jeder, der sich intensiver mit dem C64 beschäftigen will, eine Menge Informationen, Wissenswertes, Tips und Tricks, Grundlagen und Hinweise für Profis. Man merkt mit jeder Seite, daß dieses Buch von einem wirklichen Könner mit langer praktischer Erfahrung geschrieben wurde, ohne überflüssigen Ballast, konzentriert und doch an wichtigen Stellen ausführlich genug. Das

Buch hat 17 Kapitel, dazu kommt ein Anhang mit wichtigen Tabellen sowie ein ausführliches Stichwortregister. Behandelt werden das Basic des C64 und eine optimierte Programmierung in dieser Sprache. Einen breiten Teil nehmen ein die Architektur, die Hardware und die Ports des C64. Besonderer Wert wurde zudem auf die Programmierung des C64 in Maschinensprache gelegt. Daher fehlt auch eine ausführliche Beschreibung der ROM-Routinen nicht.

Dieses Buch kann jedem wärmstens empfohlen werden, der sich etwas näher mit dem C64 beschäftigen will, sei es nur in Basic oder auch in Maschinensprache. Ein Handbuch, das garantiert nicht im Regal verstaubt.

(gk/sk)

Raeto West: C64 Computer Handbuch, TeWi-Verlag, 600 Seiten, ISBN 3-921803-24-1. Preis 66 Mark

das Forum für alle Commodore-Fans

Die aktuelle Oktober-Ausgabe

Massenspeicher
Große Diskettenübersicht: Welche Qualitätsunterschiede gibt es? No-Name-Produkte kontra Markendisketten.

Monitore
Farbmonitore aller Normen im Vergleich; Grafik- und Textdarstellung auf Farbmonitoren.

C128
Neue Hard- und Softwareprodukte im Test; der C128 im professionellen Einsatz.

erhalten Sie ab 18.9.87 im Zeitschriftenhandel

Gutschein FÜR EIN KOSTENLOSES PROBEEXEMPLAR DES 64'er-MAGAZINS

JA, ich möchte »64'er«, das Magazin für Computerfans, kennenlernen. Senden Sie mir bitte die aktuellste Ausgabe kostenlos als Probeexemplar. Wenn mir »64'er« gefällt und ich es regelmäßig weiterbeziehen möchte, brauche ich nichts zu tun: Ich erhalte »64'er« dann regelmäßig frei Haus per Post und bezahle pro Jahr nur DM 78,– (Ausland auf Anfrage).

Vorname, Name

Straße/PLZ, Ort

Datum, 1. Unterschrift

Mir ist bekannt, daß ich diese Bestellung innerhalb von 8 Tagen bei der Bestelladresse widerrufen kann und bestätige dies durch meine zweite Unterschrift. Zur Wahrung der Frist genügt die rechtzeitige Absendung des Widerrufs.

64S21

Datum, 2. Unterschrift

Gutschein ausfüllen, ausschneiden, in ein Kuvert stecken oder auf eine Postkarte kleben und absenden an: Markt&Technik Verlag Aktiengesellschaft, Vertrieb, Postfach 1304, 8013 Haar

Fordern Sie mit nebenstehendem Gutschein ein kostenloses Probeheft an. Lernen Sie »64'er«, das Magazin für Computer-Fans, unverbindlich kennen.

Assembler-Programmierhilfen aus dem 64'er-Magazin

Das vor Ihnen liegende Sonderheft ist randvoll mit nützlichen Programmen und Tips zum Thema Basic und Assembler. Doch das 64'er-Magazin hat sich schon früher intensiv mit diesem Thema beschäftigt. Lesen Sie, was auch ältere Ausgaben des 64'er und der Sonderhefte zu bieten haben.

Maschinensprache ist die einzige Sprache, die ein Computer wirklich verstehen kann. Höhere Programme in Hochsprachen wie Pascal, Comal oder Basic müssen deshalb erst von geeigneten Programmen (Compiler oder Interpreter) in Maschinensprache übersetzt werden. Trotz einer großen Auswahl von unterschiedlichen Programmiersprachen ist es vorteilhaft, auch in der »niederen« Maschinensprache programmieren zu können. Denn Assembler, wie diese Sprache oft synonym bezeichnet wird, bietet einige wichtige Vorteile. Da Maschinprogramme nicht übersetzt werden müssen und ohne Umwege vom Prozessor des Computers verstanden werden, ist deren Abarbeitungsgeschwindigkeit extrem hoch (etwa 1000mal schneller als Basic) und eröffnet dem Programmierer technische Möglichkeiten, die mit einer höheren Sprache oft nicht zu realisieren wären. Der sehr beliebte Rasterzeileninterrupt des C64, wie er auch im Programm »Magic Border Beams« in diesem Sonderheft angewandt wird, ist nur ein Beispiel.

Super-Assembler-Kurs

Maschinensprache besitzt jedoch auch einige Nachteile, die so manchen Interessierten davon abgehalten haben, sich näher mit dieser ansonsten faszinierenden Sprache zu befassen. Denn das Erlernen von Assembler ist nicht ganz so einfach wie Basic.

Sich dieser Problematik bewußt, stellt die Redaktion des 64'er-Magazins nun ein neues Sonderheft zum Thema Maschinensprache vor. Viele Berichte und Tips sollen dem Assembler-Neuling den Weg zur Maschinensprache ebnen oder zumindest von großen Hindernissen befreien.

Es soll aber auch erwähnt werden, daß schon im Jahre 1985 ein Sonderheft zu diesem Thema erschien, das eine sehr gute Ergänzung zu dieser Ausgabe darstellt. Eine kleine Rückblende auf das Sonderheft 8/85 (Bild 1, links) soll zeigen, daß die Beiträge für den Maschinensprache-Freak auch heute von großer Bedeutung sein können.

Obgleich sich auch in dieser Ausgabe ausführliche Assemblerkurse befinden, müssen wir doch auf den Kurs des Sonderheftes 8/85 hinweisen, der mit seinem Umfang von über 70 Seiten bisher noch nicht übertroffen wurde. Auf einfache Weise lernt man hier alles über den C64 und die Maschinensprache seines 6510-Prozessors. Doch geht dieser Lehrgang über das Erlernen der reinen Assemblerbefehle weit hinaus: In praxisnahen Beispielen wird der zukünftige Assembler-Programmierer sicher geleitet, so daß man nach dem Studium des Kurses schon über eine gewisse Programmiererfahrung verfügt. Sie erlaubt es bereits, eigens entwickelte kleine Projekte erfolgreich in Maschinensprache abzufassen.

Lernt man zunächst die Arbeitsweise, den Mikroprozessor und den Arbeitsspeicher des C64 kennen, werden bald darauf, nach einem ausführlichen Blick auf die Register des Prozessors und einigen Rechenübungen im Binärsystem, die ersten Assemblerbefehle vorgestellt. Schritt für Schritt bemerkt man, daß anfänglich so unverständliche Fachbegriffe wie unmittelbare, absolute oder Zeropage-Adressierung wahrlich keine Alchimie sind.

Der richtige Einstieg

Leider ist man in der Maschinensprache des C64 nur imstande, Additionen und Subtraktionen durchzuführen. Andere Rechenoperationen müssen mühsam selbstentworfen werden. Als Alternative bietet der Autor des Kurses fertige Routinen für eine 16-Bit-Multiplikation und Division an, deren Funktionsweise selbstverständlich schrittweise erläutert wird. Übrigens bieten wir Ihnen auch in diesem Sonderheft solche vorgefertigten Assembler-Programm-Module für wichtige Funktionen an, damit Sie nicht jedesmal »das Rad neu erfinden« müssen.

Ist einem diese Materie vertraut, befindet man sich bereits im Lager der fortgeschrittenen Programmierer und wird von dort aus weiter in die Geheimnisse der Assemblersprache eingeweiht. Endlich erfährt man die Bedeutung des sonst unverständlichen Prozessor-Stacks und den Unterschied von Integer- und Fließkommazahlen in Maschinensprache und vieles mehr.

Daneben geben eine Vielzahl von Anwendungen dem Leser weitere Möglichkeiten, »Erfahrungspunkte« bezüglich der Programmiertechnik zu sammeln. Den krönenden Abschluß bildet die Programmierung der Echtzeituhren, die in den CIA-Bausteinen enthalten sind. Anders als die recht ungenaue »Software«-Uhr, die man mittels der Variable TI$ steuern kann, werden die Echtzeituhren von der sehr genauen Frequenz (50Hz) des Stromnetzes getaktet.

Geschwindigkeit durch Tabellen

Hat dieser Kurs Ihr Interesse geweckt, so schließt sich ein weiterer Kurs für den schon fortgeschrittenen Programmierer an. Er befaßt sich mit der optimierten Programmierung in Assembler. Er knüpft also dort an, wo der Assembler-Kurs beendet wurde. Das Ziel ist es nun, das Erlernte möglichst perfekt anzuwenden, so daß selbstgeschriebene Programme schneller ablaufen. Man kann sich beispielsweise vorstellen, daß die Berechnung des »Kotangens hyperbolicus« einer dreizehnstelligen Zahl selbst in Maschinensprache nur durch relativ langsame und komplexe Routinen erfolgen kann. Eine sehr geschätzte Alternative ist hier die Arbeit mit Tabellen, in denen die häufigsten Werte als Konstante abgelegt sind. Wie man solche Tabellen optimal gestaltet und ausnutzt, ist in einem ausführlichen Abschnitt dieses Kurses erklärt.

Doch die eben genannten Bereiche sind nur ein kleiner Teil der in diesem Kurs unternommenen Optimierungsversuche. Weitere Themen sind die Ausnutzung der Zero-

page, die richtige Programmierung von Schleifen, Selbstmodifikation von Programmen, sowie die Verwendung von Pufferspeichern und die Arbeit in der sogenannten Pass-Technik.

Hinzu kommen einige Tips, um die Ablaufgeschwindigkeit des Betriebssystems des C64 zu erhöhen, wie es zum Beispiel durch Eingriffe in den Systeminterrupt erfolgen kann. Durch Verkürzen oder gänzliches Abschalten der

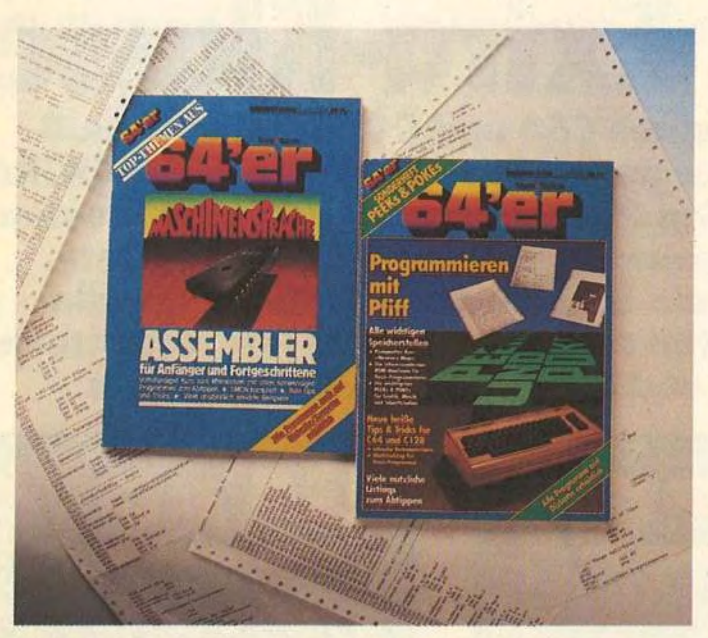


Bild 1. Das Assembler-Sonderheft 8/85 (links) und das Peeks & Pokes Sonderheft 7/86 (rechts)

internen Interruptroutine lassen sich eigene Programme erheblich beschleunigen.

Ebenso interessant dürften auch die Hinweise zur sinnvollen Nutzung des RAM-Speichers ab Adresse $E000 sein, der normalerweise nicht zur Verfügung steht, wenn der Basic-Interpreter und das Betriebssystem aktiv sind. Das Betriebssystem »liegt« förmlich über dem RAM. Inwieweit man dennoch das versteckte RAM sinnvoll für die Speicherung von Daten verwenden kann, erfahren Sie ebenfalls in diesem Kurs.

Um all die erlernten Fähigkeiten sofort praktisch ausprobieren zu können, bietet das Sonderheft 8/85 zwei fantastische Programme zum Abtippen, die sich mit der Eingabe von Assemblerprogrammen und der direkten Maschinenprogrammierung beschäftigen. Sie sind heute bereits wahre Klassiker und nicht nur unter den 64'er-Lesern bestens bekannt. Ihre Namen lauten »Hypra-Ass« und »SMON«.

Hypra-Ass ist ein Makro-Assembler und kann zu Recht als Assembler der Spitzenklasse bezeichnet werden, da er manch professionelles Programm weit hinter sich läßt. Übertroffen wird er jedoch von »Giga-Ass« seinem Nachfolger, der mit diesem Sonderheft vor Ihnen liegt.

Doch werden Sie vielleicht bemerken, daß der Begriff »Assembler« eigentlich der Name der Maschinensprache ist. Warum besitzt ein Programm den gleichen Namen? Die Verwunderung ist berechtig, läßt sich aber schnell aufklären. Computer lassen sich in der Regel nicht direkt in Assembler programmieren. Hierfür ist ein Programm als »Vermittler« notwendig, das die Assembler-Befehle in die für den Computer verständliche Form von Zahlencodes übersetzt. Denn ein Assembler-Befehl wie »LDA # $00« kann von Ihrem C64 nicht direkt entziffert werden. Er benötigt die entsprechenden Zahlencodes, die wirkliche Maschinensprache. In unserem Beispiel wären das die Bitmuster für $A9 und $00 also »10101001« und »00000000«.

Diese Aufgabe übernehmen Programme, die den gleichen Namen erhalten haben wie die Sprache, die sie übersetzen sollen: Assembler.

Auch »Giga-Ass« aus dieser Ausgabe ist eines dieser Programme und bietet dem Programmierer Möglichkeiten, die er bald nicht mehr missen möchte. Das im folgenden Gesagte gilt für beide Programme, Giga- und Hypra-Ass. Giga-Ass bietet darüber hinaus noch einiges mehr.

Zwei »Klassiker« zum Abtippen

Wie jeder gute Makro-Assembler erlaubt Hypra-Ass die Verwendung von Labels oder Variablen. Programmteile dürfen mit sogenannten Labels (Marken) versehen werden, die man durch Angabe des Label-Namens anspringen kann. Auf diese Weise kann man Unterprogrammen (Subroutines) eigene Namen geben, statt sie durch ihre abstrakte Startadressen anzusprechen. Der Befehl JRS UNTER verzweigt zum Beispiel in ein Unterprogramm, dem das Label »UNTER« zugewiesen wurde. Ähnlich verhalten sich auch die Variablen. Sie stehen für einen Wert oder eine anzusprechende Adresse. Innerhalb von Befehlen lassen sich mit diesen Variablen jegliche Berechnungen wie Addition oder Subtraktion durchführen und verleihen dem Programmierer bei der Erstellung seiner Programme mehr Flexibilität.

Wie man sehr bald bemerkt, muß in Maschinensprache ein Programm aus sehr vielen, einfach arbeitenden Assembler-Befehle zusammengesetzt werden. Benötigt man bestimmte Anweisungs-Folge sehr häufig, ist die ständige Wiederholung der Befehle bei der Eingabe bald recht lästig. Ein Unterprogramm wäre hier eine Lösung, doch sind die umständliche Unterscheidung von Unter- und Hauptprogramm sowie die verwirrenden direkten Sprünge für einige wenige Befehle sehr aufwendig. Eine bessere Lösung bieten hier die Makros.

Hypra-Ass mit Makros

Makro-Assembler gestatten es, eine Reihe von stets wiederkehrenden Befehlen unter einem Namen zusammenzufassen, der später wie ein normaler Befehl bei der Programmierung verwendet werden darf, obwohl er eigentlich nicht existiert.

Hypra-Ass besitzt in diesem Zusammenhang die sehr ungewöhnliche Trennung zwischen globalen und lokalen Variablen, wie man sie sonst nur bei höheren Programmiersprachen findet. Dies bezieht sich auf den Geltungsbereich der Variablen. Lokale Variablen, die in einem Makro definiert wurden, gelten nur dort. Globale Variablen können mit gleichem Namen durchaus außerhalb des Makros verwendet werden.

Auch das Einfügen von Tabellen in das Maschinenprogramm kann mit Hypra-Ass schnell und übersichtlich vorgenommen werden. Selbst Texte im Programm, deren Ausgabe man in der Regel Buchstabe für Buchstabe programmieren müßte, werden in ihrer Gesamtheit als Text in das Assemblerprogramm eingebunden.

Natürlich beherrscht Hypra-Ass auch die bedingte Assemblierung, die recht ungewöhnliche Programmlösungen zuläßt. Mit speziellen Kommandos werden Teile des Programmes beim späteren Übersetzungsvorgang nur dann assembliert, wenn eine bestimmte Bedingung erfüllt ist. Ansonsten kommt ein anderer Programmteil zur Übersetzung.

Daneben lassen sich innerhalb eines Programms weitere auf der Diskette befindliche Programme mit dem »Merge«-Befehl einbinden. Die Verkettung von Maschinen-

programmen und die Ausnutzung von Programmbibliotheken ist somit kein Problem mehr.

Bei der Eingabe der Assembler-Programme erweist sich Hypra-Ass als ungewöhnlich. Während andere Makro-Assembler hierfür komplexe Editorprogramme anbieten, erfolgt die Eingabe von Hypra-Ass-Programmen mit dem bekannten Basic-Editor des C64. Anhand von Zeilennummern läßt sich ein Assemblerprogramm einfach wie ein Basic-Programm schreiben und auf gewohnte Weise editiern.

Basic-Editor zur Eingabe

Hat man unter Zuhilfenahme dieser Annehmlichkeiten sein Assembler-Programm eingegeben (Bild 2 zeigt ein typisches Hypra-Ass-Programm auf dem Monitor) kann der Quelltext, wie das Programm in seiner jetzigen Form genannt wird, auf Diskette gespeichert werden. Nun tippt man wie zum Start eines Basic-Programmes den Befehl RUN, und sogleich wird das Quell-Programm in drei Durchgängen in lauffähigen Maschinencode umgewandelt, den man ebenfalls auf Diskette speichern kann oder mit SYS und der entsprechenden Startadresse sofort startet. Sollten bei der Erstellung des Programmes Eingabefehler entstanden sein, hilft Hypra-Ass mit vielen detaillierten Fehlermeldungen bei der genauen Lokalisierung der fehlerhaften Programmzeile.

Hat man sein Maschinenprogramm assembliert, liegt reiner Maschinencode vor, der nur sehr umständlich – mit einem Maschinensprache-Monitor – entziffert werden kann. Veränderungen oder die Verbesserung von Fehlern sowie das Einfügen von Befehlen lassen sich bei solchen Programmen nur mühsam oder überhaupt nicht durchführen. In diesem Fall kann man auf den noch vorhandenen Hypra-Ass-Quelltext seines Programmes zurückgreifen, dort die gewünschten Änderungen anbringen, um das Programm anschließend erneut zu assemblieren. Existiert das Quellprogramm jedoch nicht mehr, etwa weil man es unachtsam gelöscht hat, oder will man ein fremdes Maschinenprogramm umschreiben, kann die Veränderung von Maschinenprogrammen zur Tortur werden. Doch darf man nicht verzweifeln, denn passend zu Hypra-Ass befindet sich im gleichen Sonderheft ein Reassembler, der jedes Maschinenprogramm wieder in Hypra-Ass-Quelltext verwandelt. Er ist quasi das Gegenstück zu Hypra-Ass und besitzt den einfachen Namen »Reass«.

Hypra-Ass und zurück

Ein mit »Reass« erzeugter Quelltext, wie er in Bild 3 dargestellt ist, kann nun auf einfache Weise mit dem Editor von Hypra-Ass editiert und verändert werden.

Der »Reass« ist somit eine wertvolle Ergänzung zu Hypra-Ass, die man als Programmierer sehr bald zu schätzen weiß. Ebenso wichtig ist auch ein drittes interessantes Listing im Sonderheft 8/85.

SMON ist jedoch kein Assembler, sondern ein Maschinensprache-Monitor. Ein solches Programm erlaubt es, den Inhalt des Computerspeichers zu inspizieren und gegebenenfalls zu verändern. Mittels einfacher Befehle können die im Speicher des C64 enthaltenen Werte sichtbar gemacht werden (Memory-Dump). Auf diese Weise kann man zum Beispiel den Maschinencode eines Assemblerprogrammes betrachten und ihn durch direkte Eingabe der Codes manipulieren. Wesentlich bequemer ist jedoch die Eingabe von Maschinenbefehlen durch den eingebauten Mini-Assembler. Wie die Vorsilbe »Mini« bereits vermuten läßt, ist dieser Assembler bei weitem nicht so komfortabel wie Hypra-Ass. Es ist nur die Eingabe von direkten Befehlen ohne Variable und Makros gestattet, doch dürfen bis zu 30 Labels zur Markierung von Programmstellen verwendet werden.

Ein ebenso kleiner Disassembler wandelt die Codes wieder in Assembler-Befehle zurück, so daß man jeden beliebigen Speicherbereich des C64 als Maschinenprogramm betrachten kann. Daneben findet man einen Befehl zum Speichern jedes gewünschten Speicherbereichs auf Diskette oder Kassette. Ein weiteres Kommando lädt die

```
100  -.LI 1,3,0
110  -.BA $C000 ; START: SYS 49152
120  -;
130  -.GL BASIN = $FFCF
140  -.GL NUMOUT = $BDCD
150  -.GL STROUT = $AB1E
160  -;
170  -ANFANG     LDA #<(TEXT1)
180  -           LDY #>(TEXT1)
190  -           JSR STROUT
200  -;
210  -           LDX #0
220  -SCHLEIFE1  JSR BASIN
230  -           CMP #" "         ; SPACE?
240  -           BEQ SCHLEIFE1    ; DANN UE
BERLESEN
250  -           CMP #13          ; 13 = RE
TURN
260  -           BEQ WEITER1
270  -           STA STORE,X
280  -           INX
```

Bild 2. Ein Assemblerprogramm wird mit Hypra-Ass so einfach wie im Basic des C64 eingegeben

gespeicherten Abschnitte wieder an ihre richtige Adresse. Soll der Speicherbereich an eine andere Adresse geladen werden, ist auch dies für SMON kein Problem.

Doch über diese Befehle hinaus zeigt SMON erst in einer Vielzahl von zusätzlichen Anweisungen seine volle Leistungsfähigkeit.

Speicherabschnitte können problemlos verschoben werden. Werden Maschinenprogramme allerdings verschoben, sind diese an neuer Position meist nicht mehr lauffähig, da Sprünge oder Adressierungen auf den ursprünglich belegten Speicherbereich erfolgen. Um solche Programme dennoch verwenden zu können, müssen in der Regel sämtliche Befehle mit direkter Adressierung auf die neue Position im Speicher angepaßt werden. Ist diese Arbeit von Hand vorzunehmen, kann das Vorhaben zu einer nerven- und zeitraubenden Beschäftigung werden. SMON übernimmt diese Arbeit auf Wunsch automatisch und stets korrekt, so daß ein verschobenes Programm auch an der neuen Speicheradresse startbereit ist. Lediglich bei der indirekten Adressierung versagen die Künste

```
100  -eq         elf fd2=$ffd2
110  -;
120  -           .ba $8000
130  -           ;
140  -18000      ldy #$00
150  -18002      lda t18010,y
160  -           cmp #$23
170  -           beq 1800f
180  -           jsr elffd2
190  -           iny
200  -           bne 18002
210  -1800f      rts
220  -;
230  -t18010     .by $c4,$49,$45,$53,$20,
$49,$53

   >>REASS<< weist den erkannten Labels
   automatisch Werte zu (z.B. 18002)
   Tabellen werden als solche erkannt
   (Zeile 230)
```

Bild 3. Ein mit »REASS« reassembliertes Programm kann mit Hypra-Ass weiter bearbeitet werden

unseres Monitors. Sie muß nach wie vor »persönlich« angeglichen werden.

Besonders selten findet man bei einem Monitor Kommandos zum Durchsuchen des Speichers. Noch ungewöhnlicher ist die Vielfalt, die SMON hier an den Tag legt.

Hier hat der Anwender nicht nur die Möglichkeit, bestimmte Abschnitte des Speichers nach besonderen Byte-Folgen zu durchstöbern, es lassen sich vielmehr in vielen Variationen auch Befehle mit unmittelbarer, absoluter oder Zeropage-Adressierung finden. Daneben entdeckt SMON spielend relative Adressierungen oder mögliche Tabellen von Programmen, wie man in Bild 4 sehen kann. Dem »Software-Spion« sind damit keine Grenzen gesetzt.

SMON: Monitor der Spitzenklasse

Haben Sie ein Maschinenprogramm mit Hypra-Ass geschrieben und korrekt assembliert, sieht man sich nun dem größten Problem der Maschinensprache gegenüber. Arbeitet mein Programm auch so, wie ich es mir vorgestellt habe? Denn trotz so angenehmer Hilfen wie Hypra-Ass sind logische Fehler bei der Programmierung die häufigsten und meist gefürchteten Nebenwirkungen beim Programmablauf. Da Maschinensprache über keine Fehlermeldungen oder Sicherheitvorkehrungen bei Ungereimtheiten im Programm verfügt, verabschiedet sich der Computer bei Fehlprogrammierungen meist mit einem letzten »Aufschrei« (wirre Zeichen sind auf dem Bildschirm zu erkennen) oder aber leise und ohne weitere Anzeichen. Solche Abstürze sind oftmals nur durch eine Reset-Taste oder den Netzschalter des C64 zu beheben.

Auch hier zeigt sich SMON als wahrer Helfer in der Not. Diverse Trace-Modi geben dem Anwender Gelegenheit, den probeweisen Ablauf eines Maschinenprogramms gefahrlos Schritt für Schritt mitzuverfolgen (Bild 5). Dabei hat man die Wahl zwischen der Einzelschritt-Abarbeitung oder die schnelle Ausführung eines festgelegten Programm-Abschnittes mit garantiertem Stop an einer gewünschten Stelle. Fehlsprünge an undefinierte Adressen werden damit sofort ausfindig gemacht und können mit dem Disassembler korrigiert werden. Größere Verbesserungen sind jedoch vorteilhafter unter Zuhilfnahme von Reass und Hypra-Ass vorzunehmen.

Durch Druck einer bestimmten Taste widerfährt unserem leistungsfähigen Monitor eine geheimnissvolle Verwandlung. Alle bisher tadellos funktionierenden Befehle werden nicht mehr akzeptiert. Von nun an arbeitet SMON als Disketten-Monitor, der es ermöglicht, die Inhalte von Disketten direkt zu beeinflussen. Einzelne Sektoren können gelesen, deren Inhalt betrachtet und auf Wunsch geändert werden. Anschließend kann der veränderte Sektor wieder auf Diskette geschrieben werden. SMON eröffnet dem Anwender damit direkte Eingriffe in die Organisation einer Diskette, die, wenn sie gezielt erfolgen, durchaus sehr nützlich sind. Unqualifizierte Manipulationen haben jedoch meist verheerende Folgen für die weitere Lesbarkeit der Versuchsdiskette. Für eigene Testzwecke ist es daher ratsam, eine Diskette zu verwenden, die keine wichtigen Daten enthält (nicht etwa die einzige Kopie von SMON).

Damit wären die wichtigsten Eigenschaften der Grundversion von SMON erläutert. Sie haben bei dem Begriff »Grundversion« richtig gelesen, denn der modulare Aufbau von SMON erlaubt es, beliebige Erweiterungen einfach und schnell in den Monitor einzubinden. Zur Wahl stehen zum Beispiel ein erweiterter Disassembler, der die Fähigkeit besitzt, sogenannte »illegale« Assemblerbefehle zu übersetzen, oder ein wesentlich komfortablerer Disketten-

```
,c259  60        rts
----------------------------------------
,c25a  a2 a4     ldx #a4
,c25c  20 80 c2  jsr c280
,c25f  20 80 c2  jsr c280
,c262  d0 1c     bne c280
,c264  20 7e c2  jsr c27e
,c267  a9 fe     lda #fe
,c269  85 fd     sta   fd
,c26b  a9 ff     lda #ff
,c26d  85 fe     sta   fe
,c26f  20 c2 c2  jsr c2c2
,
.frc259,c000,cfff
,c24c  f0 0b     beq c259

.frc280,c000,cfff
,c262  d0 1c     bne c280
,c272  d0 0c     bne c280
.
.
.SMON durchsucht sich selbst nach
.relativen Spruengen mit dem
.FR-Befehl.
.
```

Bild 4. »SMON« als Speicher-Detektiv. Hier ist er auf der Suche nach relativen Adressen

monitor, der zusätzlich die Ausgabe beliebiger Speicherbereiche im RAM der Floppystation erlaubt.

Ebenfalls interessante SMON-Module bieten Befehle zur Ausgabe des Computerspeichers in Bit-Darstellung. SMON kann somit auch als Sprite-Editor eingesetzt werden. Ein weiteres Kommando kopiert SMON komplett in gewünschte Speicherabschnitte, so daß dieser leistungsfähige Monitor an jeder Position im Speicher einsetzbar ist. Ein spezieller Befehl kopiert den Zeichensatz des C64 in einen bearbeitbaren RAM-Bereich. Mit Hilfe der Bitdarstellung wird SMON zum wirkungsvollen Zeichensatzeditor. SMON scheint hier keine Grenzen zu kennen.

Ein guter Monitor muß trotz hoher Leistungsfähigkeit möglichst klein sein, das heißt er soll bei der Arbeit mit dem Computer nur wenig Speicherplatz belegen. Die Grundversion von SMON hält sich mit einer Größe von 4 KByte an diese Regel. Doch soll diese Größe nicht überschritten werden, finden die Erweiterungsmodule keinen Platz mehr. Aus diesem Grund müssen bei der Verwendung der neuen Befehle einige andere Kommandos weichen, so daß stets nur ein Teil der Anweisungen gleichzeitig zur Verfügung steht. Es ist deshalb empfehlenswert, spezielle SMON-Versionen mit eigens gewählten Befehlskombinationen für jeweilige Anwendungsbereiche zusammenzustellen.

SMON runderneuert

In diesem Zusammenhang müssen wir noch auf Sonderheft 12 hinweisen, das eine verbesserte und erweiterte, also quasi »runderneuerte« Version des SMON enthält. Unter dem Namen »PROMON 64« bietet der neue Monitor die Annehmlichkeiten des alten SMON mit allen hier besprochenen Erweiterungen. Weitere interessante Features, wie zum Beispiel das Kodieren von Speicherbereichen, wurden zusätzlich implementiert.

Ein Nachteil von PROMON ist seine Länge von 8 KByte, die für einen professionellen Monitor fast unzumutbar ist. Man wird jedoch durch die »geballte« Leistungsfähigkeit dieses Maschinensprache-Monitors entschädigt.

Wird allumfassende Leistung verlangt, ist PROMON 64 der geeignete Monitor. Benötigt man aber einen kleinen Helfer mit großen Fähigkeiten, etwa bei der Bearbeitung von umfangreichen Maschinenprogrammen, werden die einzelnen Spezialversionen des SMON sicherlich gute Dienste leisten.

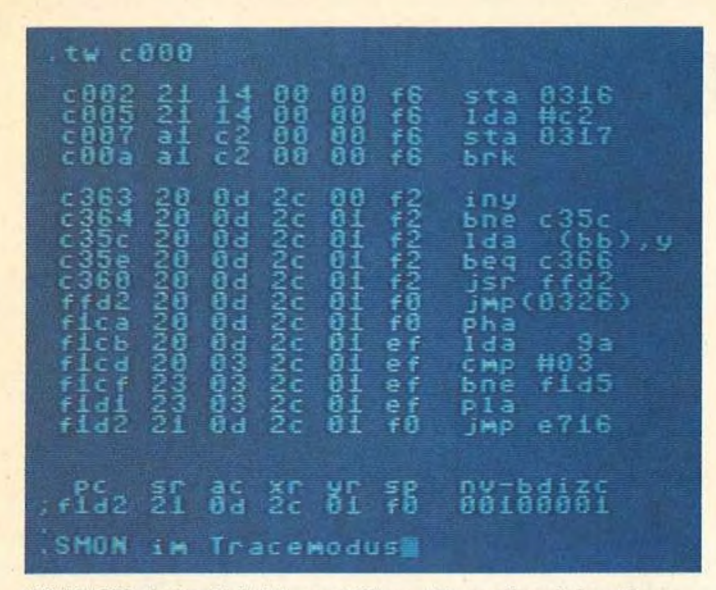


Bild 5. Nächster Befehl...und Stop...Einer der vielen Trace-Modi von SMON

Doch wenden wir uns nun ab von Assemblern und Monitoren und begeben uns wieder in die Theorie der Assembler-Programmierung. Denn es soll am Ende dieses Überblicks auch ein Kurs erwähnt werden, der sich an Programmierer von Basic sowie an Maschinensprache-Freaks richtet. Unter dem Namen »Memory-Map – mit Wandervorschlägen« erschien er im »PEEKS & POKES«-Sonderheft 7/86 (Bild 1, rechts).

Alle Speicherstellen des C64 von Adresse 0 bis Adresse 1023 – also auch die so wichtige Zeropage – sind dort ausführlich erklärt. Doch handelt es sich nicht nur um eine einfache Tabelle, die nur Kurzbeschreibungen bereitstellt. Es wird vielmehr jede Adresse gründlich inspiziert. Viele Beispiele und Tips geben dabei Auskunft, ob und in welchem Umfang diese Speicherstellen nützlich sein können. Tips für die Anwendung in Maschinensprache oder Basic fehlen natürlich auch nicht.

Speicherlandkarte für den C64

Passend zum Aufgabenbereich der jeweiligen Adressen erhält der Leser zusätzlich Wissenswertes über allgemeine Themen zum C64. Das Geheimnis der Low- und High-Bytes und die Übergabe von Basic-Werten an ein Maschinenprogramm sind nur zwei Beispiele.

Wer gemischte Programme schreiben will, das heißt Basic-Programme, die von Maschinenroutinen unterstützt werden, der wird sicherlich mit Aufmerksamkeit die Abschnitte über den Aufbau von Basic-Variablen und Feldern verfolgen. Daneben stellt sich die Zufallsfunktion RND als gar nicht so zufällig heraus. Wie man aber dennoch gutverteilte Zufallszahlen erzeugen kann, ist schrittweise in der Memory-Map erläutert.

Ein großer Teil des Kurses widmet sich der Tastatur des C64, die mehr Eigenheiten in sich birgt, als man glauben möchte. Eigenheiten, die sich besonders in Maschinenprogrammen hervorragend nutzen lassen. Auch die oft vergessene RS232-Schnittstelle und ihre Bedienung in Basic und Maschinensprache fehlen nicht in dieser geballten Sammlung der C64-Informationen.

(Michael Thomas/sk)

512 KByte am C64

Die Programmierung der Commodore-RAM-Erweiterungen war bislang nur mit dem C128 möglich. Wir zeigen Ihnen, wie Sie diese preisgünstigen Speichermodule auch an Ihrem C64 einsetzen können. Gleichzeitig erfahren Sie vieles darüber, wie diese Speichermodule in Maschinensprache anzusprechen sind.

Für alle, die an die Grenzen des C64-Speichers gestoßen sind, bietet Commodore die RAM-Erweiterungsmodule 1700 (128 KByte zusätzlich) und 1750 (512 KByte) an (Bild 1). Entwickelt wurden diese Module ursprünglich für den C128.

Leider muß der C64-Besitzer für die Benutzung dieses RAM-Moduls noch auf die GEOS-Version V1.3 warten, da zum einen die Demo-Software zur 1700/1750 nur auf dem C128 läuft und zum anderen der C64 (im Gegensatz zum C128) nicht über Befehle für die Verwaltung der Speichererweiterung verfügt. Diesen Softwaremangel wollen wir nun beheben.

Zunächst sind Sie sicher daran interessiert, die Funktionstüchtigkeit Ihrer RAM-Erweiterung gründlich zu prüfen. Auf der Demo-Diskette befindet sich ein C128-Utility namens »RAM-CHECK«, das diese Aufgabe übernimmt. Listing 1 leistet nun dasselbe auf dem C64.

Bild 1. Die beiden Speichererweiterungen 1700 und 1750

»RAM-CHECK« für den C64 erkennt nach dem Start selbständig die Version des eingesteckten Moduls, ob es sich also um ein 1700- oder 1750-Modul handelt. Dies wird durch folgende Meldung ausgedrückt:
RAM CHECK FOR 1700 EXPANSION
IS RUNNING
oder:
RAM CHECK FOR 1750 EXPANSION
IS RUNNING

Die Dauer des eigentlichen RAM-Testlaufs ergibt sich nun in Abhängigkeit von der Speichermenge, die auf Fehler zu durchsuchen ist: Die 128 KByte des 1700-Moduls werden in zirka 30 Sekunden, die 512 KByte der 1750-Erweiterung in zirka 2 Minuten gecheckt.

Während eines Testlaufs blinkt der Bildschimrahmen in allen Farben. Verläuft der Test erfolgreich, erscheint nach der angegebenen Zeit die folgende Meldung:
NO ERRORS !

Dann wird das Testprogramm nicht mehr benötigt, denn Sie haben eine korrekt arbeitende RAM-Disk zur Verfügung.

Erhalten Sie jedoch die Fehlermeldung
RAM EXPANSION ERROR IN
BANK: x ADR.: xxxxx
ist der Test mehrfach zu wiederholen. Treten dabei weitere Fehler auf, sollten Sie Ihren Händler aufsuchen. Verläuft dort ein Test auf einem C128 mit dem RAM-CHECK-Programm der Commodore-Demodiskette positiv, so wird Ihr C64 mit ziemlicher Sicherheit nur Schwierigkeiten mit der Stromversorgung haben, was bei einigen älteren Ausführungen vorkommt. Ein neues Netzteil oder eine externe Stromversorgung für die RAM-Disk beseitigen diese Schwierigkeiten.

Sollten jedoch auch diese Versuche fehlschlagen, ist Ihr Modul mit hoher Wahrscheinlichkeit schadhaft und muß umgetauscht werden.

RAM-Steuerung auch in Basic

Das Basic 7.0 des C128 verfügt über drei Befehle, die speziell für den Umgang mit den RAM-Erweiterungen 1700 und 1750 entwickelt wurden. Damit auch Sie als C64-Benutzer in den Genuß dieser Befehle kommen, erweitert unser Programm »DMA-BASIC« das Basic des C64 um diese drei Befehle sowie eine weitere, C64-spezifische Anweisung. Auf der Programmservice-Diskette befindet sich zusätzlich eine brennfähige Betriebssystemversion von DMA-BASIC, die anstelle des Original-Kernel eingesetzt werden kann. Diese kann aus Platzgründen leider nicht abgedruckt werden.

Die Basic-Erweiterung »DMA-BASIC V2« (Listing 2) wird mit dem MSE eingegeben und wie ein Basic-Programm geladen und gestartet. Danach erscheint sofort eine modifizierte Einschaltmeldung. DMA-BASIC befindet sich dann ab Adresse 51200 ($C800) im Speicher und bleibt bis zur Auslösung eines Reset aktiv. Nach einem Reset kann, falls notwendig, die Erweiterung mit SYS 51200 wieder reaktiviert werden.

DMA-BASIC stellt vier Befehle zur Verfügung:

STASH speichert Daten im RAM-Erweiterungsmodul
FETCH überträgt Daten aus dem RAM-Modul in den C64-Hauptspeicher
SWAP vertauscht Daten des RAM-Moduls mit Daten aus dem C64-Hauptspeicher
BANK ermöglicht es, Daten aus dem RAM »unter« dem Basic-ROM, I/O-Bereich oder Kernel-ROM in Kombination mit einem der drei anderen Befehle zu übertragen.

Dabei sind die Befehle in der Parameter-Übergabe voll zu den gleichlautenden Kommandos des C128 kompatibel, benutzen jedoch andere Tokens und selbstverständlich andere Routinen.

Die Befehle STASH, FETCH und SWAP haben jeweils eine identische Parameterfolge:
FETCH
STASH BYTES, C64ADR, RAMADR, RAMBNK
SWAP
Die Bedeutungen dieser Parameter sind folgende:

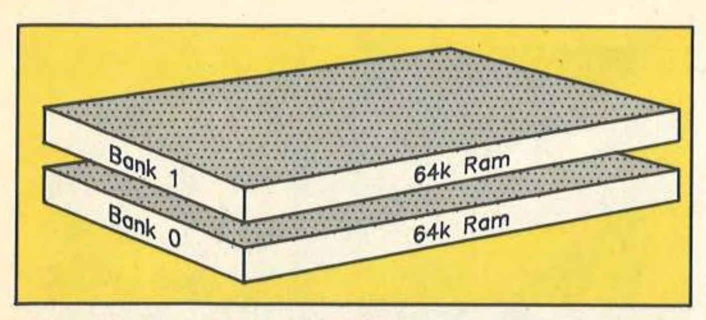


Bild 2. Die Organisation der RAM-Erweiterung 1700

- BYTES gibt an, wie viele Bytes übertragen werden sollen
- C64ADR ist die Startadresse der Datenübertragung im C64-Hauptspeicher (0-65535)
- RAMADR ist die Zieladresse der Datenübertragung im RAM-Modul (0-65535)
- RAMBNK ist die Nummer der gewünschten RAM-Erweiterungsbank (bei 1700: 0 oder 1; bei 1750: 0 bis 7).

Die Parameter sind somit Integerzahlen oder Variablen. Durch die Verwendung der Befehle werden Daten aus einem RAM-Bereich in einen anderen übertragen. Befinden sich im gewählten Bereich noch Programme oder irgendwelche anderen Daten, werden diese automatisch überschrieben beziehungsweise bei SWAP vertauscht.

Der Befehl BANK stellt eine Besonderheit dar und wurde nicht dem Basic 7.0 des C128 nachempfunden. Er erlaubt es, durch Angabe eines einzigen Parameters die RAM-Bereiche »unter« dem Basic- oder Kernel-ROM mit weiteren Transfer-Befehlen (STASH, FETCH oder SWAP) anzusprechen.

BANK 1 stellt dabei den Originalzustand des C64 her, das heißt, alle ROMs und I/O-Bereiche sind eingeblendet. Dieser Zustand wird auch nach der Ausführung oder einer Fehleingabe eines Transfer-Befehls automatisch hergestellt.

BANK 0 hingegen schaltet den kompletten Speicher auf RAM. Dazu müssen alle Systeminterrupts »verboten« werden, um einen Absturz des Computers zu verhindern.

Eine Einschränkung wollen wir nicht verheimlichen: Das Zeichensatz-ROM des C64 ist nicht übertragbar (außer in Maschinensprache).

Programmbeispiele:

Zum Ausprobieren des ersten Beispiels müssen Sie zunächst den Bildschirm durch <SHIFT CLR/HOME> löschen. Danach schreiben Sie in die oberste Bildschirmzeile einige beliebige Zeichen wie etwa »DMA-BASIC TEST 1987 BY T.POHL«. Nun bewegen Sie den Cursor tiefer und geben in eine freie Zeile folgenden Befehl ein:

```
STASH 40,1024,0,0
```

Der Computer hat nun die erste Zeile des Bildschirms auf die RAM-Erweiterung ab Adresse 0 in Bank 0 abgelegt. Die Daten bleiben solange in der RAM-Erweiterung, bis der Computer ausgeschaltet oder der Speicherbereich überschrieben wird. Ein Reset hat keinerlei Wirkung auf die abgelegten Daten.

Löschen Sie zur weiteren Demonstration erneut den Bildschirm und geben Sie

```
FETCH 40,1024,0,0
```

ein. Augenblicklich erscheint wieder die ursprüngliche Zeile. In einem kleinen Programm zusammengefaßt, sieht dieses Beispiel folgendermaßen aus:

```
10 PRINT CHR$(147);"DMA-BASIC TEST 1987 BY T.POHL"
20 FOR Y=24 TO 0 STEP -1
30 PRINT CHR$(147)
40 FETCH 40,1024+(Y*40),0,0
50 NEXT Y
60 END
```

Das Programm enthält eine Variante, welche die Textzeile von unten nach oben wandern läßt. Sie können das Programm auch verlangsamen, indem Sie folgende Veränderungen vornehmen:

```
45 FOR T=0 TO 1000
50 NEXT T,Y
```

Grundlagen für die Programmierung:

Da ein 8-Bit-Prozessor wie der 6510 des C64 nur 65536 verschiedene Speicheradressen ansprechen kann, muß ein größerer Speicherbereich in Speicherbänke eingeteilt werden. Die Bilder 2 und 3 zeigen diese Einteilung für die beiden Erweiterungen 1700 und 1750.

Die vorgestellten Routinen bedienen sich eines DMA (daher »DMA-BASIC«). DMA bedeutet »Direct Memory Access«, was man mit »direkter Speicherzugriff« übersetzt.

Ein DMA kann auf verschiedene Arten herbeigeführt werden, die alle im Handbuch der Erweiterungsmodule

Adresse	Bits	Funktion
$df00	0 - 7	Bit 4: Modulgröße (0 = 128 KByte; 1 = 512 KByte)
$df01	0 - 7	Kommando-Register für STASH: %10000100 FETCH: %10000101 SWAP: %10000110
$df02	0 - 7	Startadresse im C64 (Low-Byte)
$df03	0 - 7	Startadresse im C64 (High-Byte)
$df04	0 - 7	Startadresse in RAM-Erweiterung (Low-Byte)
$df05	0 - 7	Startadresse in RAM-Erweiterung (High-Byte)
$df06	0 - 7	Nummer der RAM-Erweiterungsbank (bei 1700: 0 - 1; bei 1750: 0 - 7)
$df07	0 - 7	Anzahl der zu übertragenden Bytes (Low-Byte)
$df08	0 - 7	Anzahl der zu übertragenden Bytes (High-Byte)

Tabelle 1. Die wichtigsten I/O-Register für DMAs

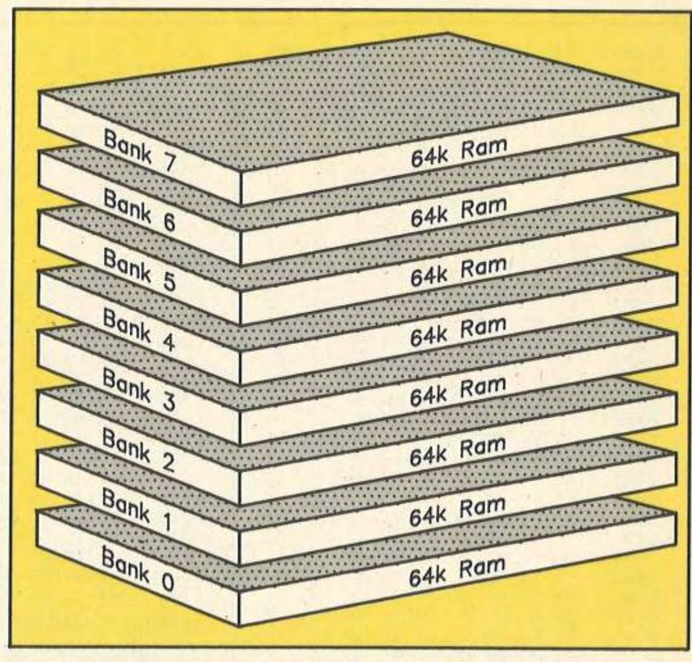


Bild 3. Die Organisation der RAM-Erweiterung 1750

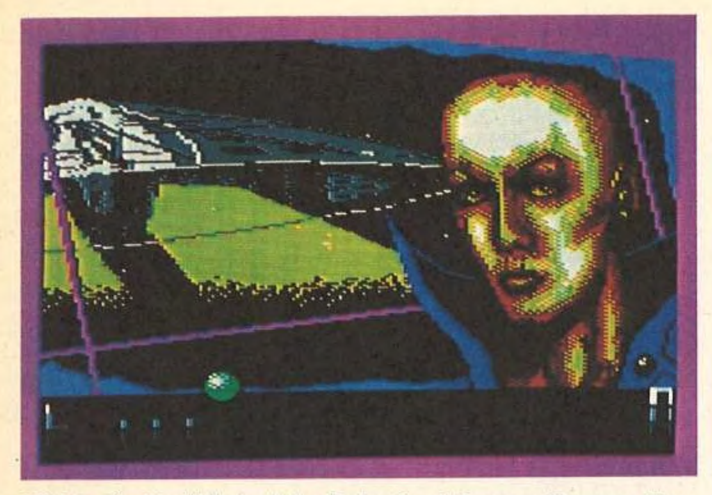


Bild 4. Ein Grafikbeispiel mit der Speichererweiterung, hier als Standbild fotografiert

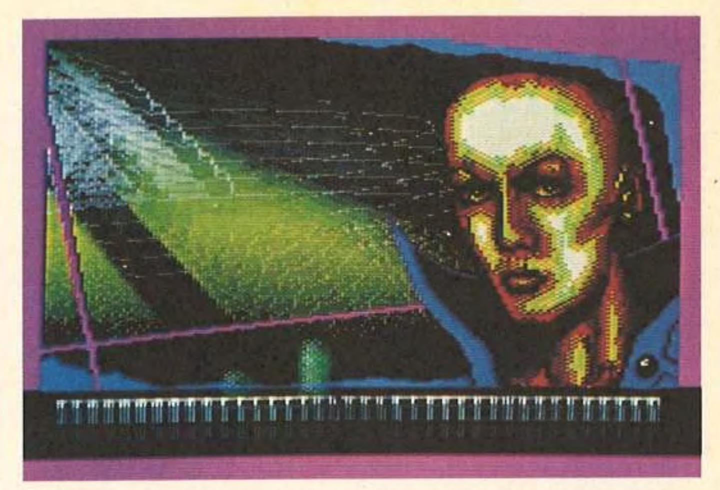
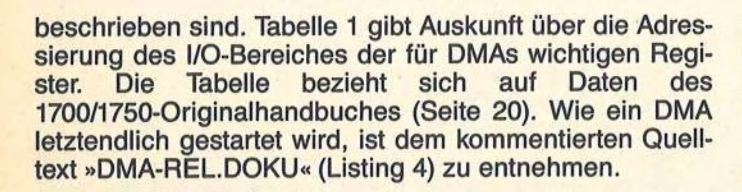


Bild 5. Der Hintergrund aus Bild 4 wird mit 32 Bildern pro Sekunde durchgeschaltet (belichtet mit ¼ Sekunde).



beschrieben sind. Tabelle 1 gibt Auskunft über die Adressierung des I/O-Bereiches der für DMAs wichtigen Register. Die Tabelle bezieht sich auf Daten des 1700/1750-Originalhandbuches (Seite 20). Wie ein DMA letztendlich gestartet wird, ist dem kommentierten Quelltext »DMA-REL.DOKU« (Listing 4) zu entnehmen.

Verschiebbare Hilfsroutinen:

DMA-Basic ist sicherlich die komfortabelste Möglichkeit für die RAM-Steuerung in Basic. Aber für manche Zwecke sind frei verschiebliche Routinen, die über SYS aufgerufen werden, praktischer.

»DMA-REL.« (Listing 3, einzugeben mit dem MSE) stellt die Transfer-Routinen von DMA-BASIC über SYS-Aufrufe zur Verfügung. Das Programm ist frei im Speicher verschiebbar, es stützt sich also nicht auf JMP/JSR-Befehle oder irgendwelche Datenfelder. Mit einem Maschinensprache-Monitor kann es ohne Adreßumrechnung fast beliebig im C64-Speicher verschoben werden. Statt der Basic-Befehle hat »DMA-REL.« folgende Syntax:

SYS Ladeadresse+X, BYTES, C64ADR, RAMADR, RAMBNK, BANK

Die Ladeadresse ist bekanntlich beliebig, im Urzustand lautet sie 51200. Der Offset »X« (hier 0, 3 und 6) dient zur Ermittlung der unterschiedlichen Einsprungadressen:

STASH: SYS Ladeadresse+0, <Parameter>
FETCH: SYS Ladeadresse+3, <Parameter>
SWAP: SYS Ladeadresse+6, <Parameter>

Die Parameter wurden, wie Sie oben gesehen haben, um BANK erweitert. Dieser ist auch nichts Neues, da er mit dem Parameter des BANK-Befehls aus DMA-BASIC identisch ist.

Noch ein Hinweis: Die Routinen aus »DMA-REL.« können, wenn das Programm an seine Ursprungsadresse geladen wird, nicht mit DMA-BASIC gleichzeitig im Speicher stehen.

Maschinenprogrammierer können aus dem kommentierten Quelltextes (Listing 4) weitere Informationen zur Funktionsweise des Programms entnehmen. Ein interessantes Anwendungsbeispiel finden Sie in den Bildern 4 und 5. In Bild 5 können Sie ansatzweise erkennen wie die gesamte Hintergrundgrafik mit 32 Bildern pro Sekunde durchgeschaltet wird. Für das menschliche Auge ergibt sich dabei eine schnelle kontinuierliche Bewegung.

(Thomas Pohl/Florian Müller)

1700 RAM-Expansion (128 KByte Speicher zusätzlich): 198 Mark
1750 RAM-Expansion (512 KByte Speicher zusätzlich): 298 Mark
Der Vertrieb der RAM-Erweiterungen erfolgt durch lizenzierte Vertragshändler sowie Versandhäuser.

```
Name : ram-check              0801 09d5
----------------------------------------
0801 : 0c 08 c3 07 9e 20 32 30  f7
0809 : 36 34 00 00 00 ea ea ea  32
0811 : a2 00 bd e0 08 20 d2 ff  0c
0819 : e8 e0 14 d0 f5 a9 00 8d  58
0821 : 00 df ad 00 df 29 10 d0  a5
0829 : 15 a9 02 8d 54 09 a2 00  5d
0831 : bd f4 08 20 d2 ff e8 e0  01
0839 : 04 d0 f5 4c 51 08 a9 08  b8
0841 : 8d 54 09 a2 00 bd f8 08  71
0849 : 20 d2 ff e8 e0 04 d0 f5  4d
0851 : a2 00 bd fc 08 20 d2 ff  cf
0859 : e8 e0 1e d0 f5 a9 00 85  0b
0861 : fc 85 fd 85 02 78 a9 15  05
0869 : 8d c4 08 20 92 08 a9 00  6f
0871 : 8d c4 08 20 95 08 ad c4  40
0879 : 08 c9 15 f0 03 4c 69 09  14
0881 : ee 20 d0 20 c5 08 b0 03  1d
0889 : 4c 55 09 4c 67 08 4c 69  06
0891 : 09 a9 84 2c a9 85 a0 01  61
0899 : a2 00 8c 07 df 8e 08 df  92
08a1 : a0 c4 a2 08 8c 02 df 8e  c2
08a9 : 03 df a4 fc a6 fd 8c 04  f9
08b1 : df 8e 05 df a6 02 8e 06  d5
08b9 : df a8 a9 00 8c 01 df 8d  c2
08c1 : 00 ff 60 00 e6 fc d0 13  99
08c9 : e6 fd d0 0f a9 00 85 fc  6e
08d1 : 85 fd e6 02 a5 02 cd 54  99
08d9 : 09 b0 02 38 60 18 60 9a  3f
08e1 : 93 11 2e 2e 2e 20 52 41  fe
08e9 : 4d 43 48 45 43 4b 20 46  2e
08f1 : 4f 52 20 31 37 30 30 31  b0
08f9 : 37 35 30 20 45 58 50 41  b6
0901 : 4e 53 49 4f 4e 0d 11 20  07
0909 : 20 20 20 49 53 20 52 55  95
0911 : 4e 4e 49 4e 47 20 2e 2e  2d
0919 : 2e 0d 11 11 4e 4f 20 45  9f
0921 : 52 52 4f 52 53 20 21 11  97
0929 : 11 0d 11 11 52 41 4d 20  cc
0931 : 45 58 50 41 4e 53 49 4f  22
0939 : 4e 20 20 45 52 52 4f 52  e2
0941 : 20 21 21 21 0d 11 42 41  43
0949 : 4e 4b 3a 20 20 41 44 52  91
0951 : 2e 3a 20 00 a2 00 bd 1a  fa
0959 : 09 20 d2 ff e8 e0 10 d0  9e
0961 : f5 58 a9 0e 8d 20 d0 60  8c
0969 : a2 00 bd 2a 09 20 d2 ff  9d
0971 : e8 e0 23 d0 f5 a6 02 a9  9c
0979 : 00 20 cd bd a2 00 bd 4d  70
0981 : 09 20 d2 ff e8 e0 07 d0  a2
0989 : f5 a6 fc a5 fd 20 cd bd  59
0991 : 58 a9 0e 8d 20 d0 a9 11  44
0999 : 4c d2 ff 28 43 29 20 44  da
09a1 : 4d 41 2d 56 45 52 53 49  6c
09a9 : 4f 4e 20 31 39 38 37 20  c0
09b1 : 42 59 20 54 2e 50 4f 48  66
09b9 : 4c 20 45 53 20 4b 41 4e  cf
09c1 : 4e 20 4e 55 52 20 45 49  2b
09c9 : 4e 45 4e 20 47 45 42 45  84
09d1 : 4e 20 21 00 20 06 b2 a4  be
```

Listing 1. »RAM-CHECK« prüft Ihre Speichererweiterung

```
Name : dma-basic v2           0801 0b46
----------------------------------------
0801 : 0c 08 c3 07 9e 20 32 30 . f7
0809 : 36 34 00 00 00 ea ea ea  32
0811 : a9 40 a2 08 85 fb 86 fc  d0
0819 : a9 00 a2 c8 85 fd 86 fe  e4
0821 : a0 00 b1 fb 91 fd e6 fb  4a
0829 : d0 02 e6 fc e6 fd d0 02  f9
0831 : e6 fe a5 fd c9 06 a5 fe  21
0839 : e9 cb 90 e6 4c 00 c8 20  31
0841 : a0 e5 78 20 15 fd 58 a0  da
0849 : 00 b9 34 c8 20 d2 ff c8  76
0851 : c0 93 d0 f5 a9 c8 a2 c8  cb
0859 : 8d 04 03 8e 05 03 a9 8b  a1
0861 : a2 c9 8d 06 03 8e 07 03  d3
0869 : a9 c0 a0 c9 8d 08 03 8c  12
0871 : 09 03 60 9a 93 11 1d 1d  d8
0879 : 20 20 20 20 20 43 4f 4d  a9
0881 : 4d 4f 44 4f 52 45 20 36  ad
0889 : 34 20 42 41 53 49 43 20  53
0891 : 56 32 20 2b 44 4d 41 0d  3c
0899 : 1d 1d 31 39 32 2f 35 37  98
08a1 : 36 4b 20 52 41 4d 20 53  75
08a9 : 59 53 54 45 4d 20 33 38  7d
08b1 : 39 31 31 20 42 59 54 45  9e
08b9 : 53 20 46 52 45 45 0d 1d  e5
08c1 : 1d 1d 1d 1d 28 43 29 20  d9
```

Listing 2. »DMA-BASIC V2« stellt die Befehle STASH, FETCH, SWAP und BANK zur Verfügung

```
08c9 : 43 4f 4d 4d 4f 44 4f 52  aa
08d1 : 45 20 45 4c 45 43 54 52  66
08d9 : 4f 4e 49 43 53 2c 20 4c  ba
08e1 : 54 44 2e 0d 1d 1d 1d 1d  ee
08e9 : 1d 1d 1d 1d 28 43 29 20  01
08f1 : 44 4d 41 2d 56 45 52 53  51
08f9 : 49 4f 4e 20 42 59 20 54  9a
0901 : 2e 50 4f 48 4c 11 00 a6  cf
0909 : 7a a0 04 84 0f bd 00 02  48
0911 : 10 07 c9 ff f0 3e e8 d0  5d
0919 : f4 c9 20 f0 37 85 08 c9  6b
0921 : 22 f0 55 24 0f 70 2d c9  52
0929 : 3f d0 04 a9 99 d0 25 c9  4f
0931 : 30 90 04 c9 3c 90 1d 84  a9
0939 : 71 a0 00 84 0b 88 86 7a  8f
0941 : ca c8 e8 bd 00 02 38 f9  46
0949 : 9e a0 f0 f5 c9 80 d0 2f  74
0951 : 05 0b a4 71 e8 c8 99 fb  66
0959 : 01 c9 00 f0 38 38 e9 3a  be
0961 : f0 04 c9 49 d0 02 85 0f  40
0969 : 38 e9 55 d0 a0 85 08 bd  d7
0971 : 00 02 f0 e0 c5 08 f0 dc  e5
0979 : c8 99 fb 01 e8 d0 f0 a6  53
0981 : 7a e6 0b c8 b9 9d a0 10  75
0989 : fa b9 9e a0 d0 b5 f0 0f  b8
0991 : bd 00 02 10 bd 99 fd 01  74
0999 : c6 7b a9 ff 85 7a 60 a0  76
09a1 : 00 b9 ef c9 d0 02 c8 e8  c5
09a9 : bd 00 02 38 f9 ef c9 f0  16
09b1 : f5 c9 80 d0 04 05 0b d0  fb
09b9 : 99 a6 7a e6 0b c8 b9 ee  dc
09c1 : c9 10 fa b9 ef c9 d0 e0  da
09c9 : f0 c6 10 0f 24 0f 30 0b  94
09d1 : c9 ff f0 07 c9 cc b0 06  89
09d9 : 4c 24 a7 4c f3 a6 38 e9  d4
09e1 : cb aa 84 49 a0 ff ca f0  63
09e9 : 08 c8 b9 ef c9 10 fa 30  2b
09f1 : f5 c8 b9 ef c9 30 05 20  29
09f9 : 47 ab d0 f5 4c ef a6 20  28
0a01 : 73 00 20 c9 c9 4c ae a7  bf
0a09 : c9 cc 90 04 c9 d1 90 06  56
0a11 : 20 79 00 4c ed a7 38 e9  48
0a19 : cc 0a aa bd e8 c9 48 bd  c6
0a21 : e7 c9 48 4c 73 00 4b ca  82
0a29 : 60 ca 75 ca 8a ca 53 54  9a
0a31 : 41 53 c8 46 45 54 43 c8  ac
0a39 : 53 57 41 d0 42 41 4e cb  a1
0a41 : 00 a9 00 8d 00 df ad 00  7d
0a49 : df 29 10 d0 03 a9 02 2c  b9
0a51 : a9 08 8d 4b ca 60 20 8a  10
0a59 : ad 20 f7 b7 a5 14 8d 07  4a
0a61 : df a5 15 8d 08 df 20 fd  06
0a69 : ae 20 8a ad 20 f7 b7 a5  6b
0a71 : 14 8d 02 df a5 15 8d 03  08
0a79 : df 20 fd ae 20 eb b7 a5  49
0a81 : 14 8d 04 df a5 15 8d 05  9c
0a89 : df 60 ff 20 02 ca 20 17  c1
0a91 : ca ec 4b ca 90 05 a2 10  d9
0a99 : 4c 8b e3 a0 84 4c a1 ca  7e
0aa1 : 20 02 ca 20 17 ca ec 4b  8b
0aa9 : ca 90 05 a2 10 4c 8b e3  aa
0ab1 : a0 85 4c a1 ca 20 02 ca  a6
0ab9 : 20 17 ca ec 4b ca 90 05  0c
0ac1 : a2 10 4c 8b e3 a0 86 4c  e6
0ac9 : a1 ca 20 9e b7 e0 02 90  57
0ad1 : 0a a2 01 8e a0 ca a2 0e  45
0ad9 : 4c 8b e3 8e a0 ca 60 01  99
0ae1 : 8e 06 df ad a0 ca d0 18  f4
0ae9 : 8c 01 df 78 a5 01 48 a9  d4
0af1 : 00 85 01 8d 00 ff 68 85  53
0af9 : 01 a9 01 8d a0 ca 58 60  43
0b01 : a9 00 8c 01 df 8d 00 ff  58
0b09 : 60 28 43 29 20 44 4d 41  4f
0b11 : 2d 56 45 52 53 49 4f 4e  5e
0b19 : 20 31 39 38 37 20 42 59  57
0b21 : 20 54 2e 50 4f 48 4c 20  aa
0b29 : 45 53 20 4b 41 4e 4e 20  89
0b31 : 4e 55 52 20 45 49 4e 45  25
0b39 : 4e 20 47 45 42 45 4e 20  da
0b41 : 21 00 00 00 21 a0 2e 8e  4f
```

Listing 2. (Schluß)

```
Name : dma-rel.                  c800 c8db
------------------------------------------
c800 : a9 84 2c a9 85 2c a9 86  99
c808 : 48 a9 00 8d 00 df ad 00  8c
c810 : df 29 10 d0 03 a9 02 2c  80
c818 : a9 08 8d 06 df 20 fd ae  3e
c820 : 20 8a ad 20 f7 b7 a5 14  f1
c828 : 8d 07 df a5 15 8d 08 df  83
c830 : 20 fd ae 20 8a ad 20 f7  85
c838 : b7 a5 14 8d 02 df a5 15  58
c840 : 8d 03 df 20 fd ae 20 eb  f8
c848 : b7 a5 14 8d 04 df a5 15  89
c850 : 8d 05 df ec 06 df 90 06  a3
c858 : 68 a2 10 4c 8b e3 8e 06  bd
c860 : df 20 fd ae 20 9e b7 e0  3c
c868 : 02 90 0a a2 01 86 02 68  a6
c870 : a2 0e 4c 8b e3 86 02 68  e9
c878 : a8 a5 02 d0 17 8c 01 df  27
c880 : 78 a5 01 48 a9 00 85 01  c7
c888 : 8d 00 ff 68 85 01 a9 01  2b
c890 : 85 02 58 60 a9 00 8c 01  07
c898 : df 8d 00 ff 60 28 43 29  e4
c8a0 : 20 44 4d 41 2d 56 45 52  9d
c8a8 : 53 49 4f 4e 20 31 39 38  1e
c8b0 : 37 20 42 59 20 54 2e 50  b1
c8b8 : 4f 48 4c 20 2e 2e 2e 45  da
c8c0 : 53 20 4b 41 4e 4e 20 4e  93
c8c8 : 55 52 20 45 49 4e 45 4e  b0
c8d0 : 20 47 45 42 45 4e 20 21  b7
c8d8 : 00 00 00 00 00 00 00 00  d9
```

Listing 3. »DMA-REL.« stellt über SYS aufrufbare Transfer-Routinen aus »DMA-BASIC V2« zur Verfügung


```
0 ;
1 ;
2 ;              dokumentiertes quell-code listing zu
3 ;
4 ; dma-rel. (relokatibel = kann ohne adressen-umrechnung verschoben
5 ;                         werden)
6 ;
9 ;
10 * = 51200
10000 ;
10001 ;flaggen definieren
10002 ;
10003 bankflg = $df06
10004 flgge   = 2
10005 ;
10006 ;durch variablen einsprung --> uebertragungs-art waehlen
10007 ;naehere beschreibung der befehle siehe anleitung !
10008 ;
10009 stash lda #%10000100   ;uebertragungsart waehlen
10010 .byt $2c               ;skip (nachfolgenden befehl ueberspringen)
10011 fetch lda #%10000101   ;-"-
10012 .byt $2c               ;skip
10013 swap  lda #%10000110   ;-"-
10014 ;
10015 pha                    ;nummer des befehls merken
10016 ;
10017 ;
10018 ;test auf ram-disk-version
10019 ;
10020 chdisk lda #0          ;status-register
10021        sta $df00       ;loeschen
10022        lda $df00       ;und relevantes bit testen
10023        and #16         ;
10024        bne skiper      ;
10025        lda #2          ;1700 (128k. 2 baenke)
10026 .byt $2c               ;skip
10027 skiper lda #8          ;1750 (512k. 8 baenke)
10028        sta bankflg     ;maximale bank-anzahl merken
10029 ;
10030 ;uebertragungswerte holen
10031 ;
10032 werget jsr $aefd       ;teste auf komma
10033        jsr $ad8a       ;hole anzahl bytes
10034        jsr $b7f7       ;konvertiere in adressformat
10035        lda $14         ;setze anzahl der zu
10036        sta $df07       ;uebertragenden bytes
10037        lda $15         ;in i/o register
10038        sta $df08       ;-"-
10039 ;
10040        jsr $aefd       ;teste auf komma
10041        jsr $ad8a       ;hole startadresse im c64
10042        jsr $b7f7       ;konvertiere in adress-format
10043        lda $14         ;setze startadresse
10044        sta $df02       ;in i/o register
10045        lda $15         ;-"-
10046        sta $df03       ;-"-
10047 ;
10048        jsr $aefd       ;teste auf komma
10049        jsr $b7eb       ;hole startadresse und banknummer
10050        lda $14         ;der ram-erweiterung
10051        sta $df04       ;setze in i/o register
10052        lda $15         ;-"-
10053        sta $df05       ;-"-
10054 ;
10055 cont   cpx bankflg     ;pruefe, ob banknummer mit der
10056        bcc skips2      ;tatsaechlichen zahl uebereinstimmt
10057 ;
10058        pla             ;nein, dann fehler
10059        ldx #16         ;"out of memory ..."
10060        jmp $e38b       ;-"-
10061 ;
10062 skips2 stx $df06       ;ok, dann in i/o register
10063        jsr $aefd       ;teste auf komma
10064        jsr $b79e       ;bank-option ins x-register
10065        cpx #2          ;falsche bank-option "?"
10066        bcc skup        ;nein, flagge setzen
10067        ldx #1          ;ja, flagge auf standard
10068        stx flgge       ;-"-
10069        pla             ;-"-
10070        ldx #14         ;und fehler
10071        jmp $e38b       ;"illegal quantity ..."
10072 ;
10073 skup   stx flgge       ;flagge setzen
10074        pla             ;nummer des befehls zurueckholen
10075        tay             ;und ins y-register
10076        lda flgge       ;flagge testen
10077        bne rom         ;rom-version
10078 ;
10079 ram    sty $df01       ;ram-version, befehlsnummer setzen
10080        sei             ;interrupt sperren
10081        lda $01         ;alte speicher-konfiguration
10082        pha             ;merken
10083        lda #0          ;alles auf ram
10084        sta $01         ;schalten
10085        sta $ff00       ;dma mit 'dummy' starten
10086        pla             ;alte konfiguration zurueckholen
10087        sta $01         ;und setzen
10088        lda #1          ;bank-option auf
10089        sta flgge       ;standard
10090        cli             ;interrupt wieder zulassen
10091 rts                    ;zurueck ins basic
10092 ;
10093 rom    lda #0          ;rom-version
10094        sty $df01       ;befehls-nummer setzen
10095        sta $ff00       ;und mit 'dummy' starten
10096 rts                    ;zurueck ins basic
10097 ;
10098 ;
10099 .asc "(c) dma-version" ;copyright-vermerk
10100 .asc " 1987 by t.pohl"
10101 .byt 0,0,0
10102 .end
```

Listing 4. »DMA-REL. DOKU« ist der kommentierte Quelltext zu Listing 3

Jetzt wird's bunt

Dieses Programm bringt Farbe in Ihre Grafik-Hardcopies, auch wenn Sie keinen Farbdrucker besitzen. Der Trick dabei sind mehrere Farbbänder, die nacheinander verwendet werden.

Heutzutage ist es nichts Besonderes mehr, Computergrafiken auf einem Drucker auszudrucken. Durch die passende Software erhält man dabei Ausdrucke mit bis zu zwölf verschiedenen Graustufen. Wollte man farbige Ausdrucke erzeugen, scheiterte dies meistens an der fehlenden Farbfähigkeit der Drucker.

Wenn man aber einen Farbdrucker besaß, ergaben sich ähnliche Probleme, denn die meiste Software ist auf Einfarbdrucker ausgelegt.

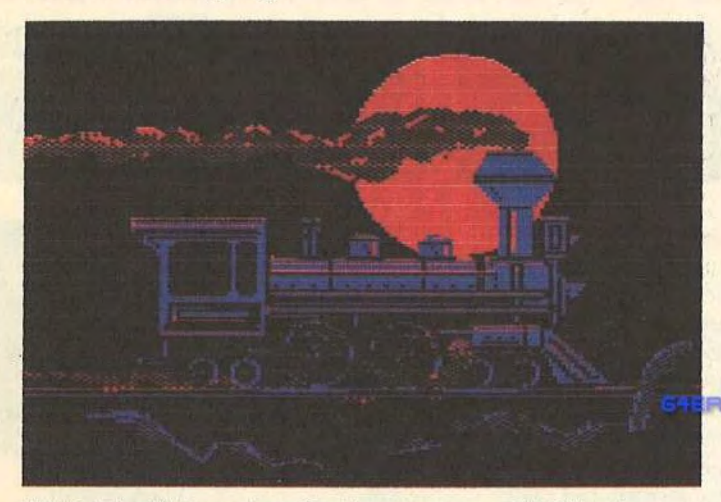


Bild 1. Ein Bild aus dem Grafikprogramm »Paint Magic«

Dieses Druckprogramm bietet hier eine Alternative. Mit bis zu vier Farbbändern werden hintereinander die einzelnen Farben gedruckt. Das Ergebnis ist eine mehrfarbige Grafik erster Klasse. Zwei Kostproben sehen Sie in Bild 1 und 2 (gedruckt mit einem Epson FX-85).

Das Programm läuft mit allen Epson-kompatiblen Druckern, des weiteren problemlos auf einem MPS 802 mit unserem SUPER-ROM (64'er 1/1987). Die Anpassung an andere Drucker ist mit der menügesteuerten Druckeranpassung kein Problem. Für Besitzer eines MPS 801/803 gibt es ebenfalls eine Druckerroutine (Listing 2).

Um eine solche Grafik auszudrucken, müssen Sie Listing 1 mit dem MSE eingeben und speichern. Interessierte finden auf der Programmservice-Diskette den dokumentierten Quellcode im Hypra-Ass-Format.

Nach dem Start erscheint die Grafikseite, in die Sie nun mit <L> eine Grafik laden. Wenn die Grafik verschoben ist, rückt man sie durch <DEL> zurecht. Die Tasten <+> und <-> ändern die Schrittweite der Verschiebung. Die Größe des Schrittes sieht man auf der Hilfstafel (Taste <H>), wo alle Befehle kurz aufgelistet sind.

Mit den vier Funktionstasten verändert man die Farben der Originalgrafik. (Mit <SHIFT> erniedrigt man den Farbwert, ohne <SHIFT> erhöht man ihn.)

Mit den Tasten <1> bis <4> filtert man jeweils einen Farbanteil aus der Grafik. Das gefilterte Bild wird daraufhin auf der zweiten Grafikseite dargestellt.

Mit der Taste <P> drucken Sie nun den ersten Farbanteil. Danach müssen Sie das Papier wieder an die Ausgangsposition zurückdrehen. Dabei empfiehlt es sich, als Markierung eine horizontale Linie an der Papierführung anzubringen und auf dem Papier weiterzuführen (am besten mit Tipp-Ex), um dann genau zurückdrehen zu können. Manche Drucker bewerkstelligen dies automatisch (muß vorher im Druckermenü eingestellt werden).

Nun wiederholt man entweder den gleichen Ausdruck, um sattere Farben zu erreichen oder man filtert einen neuen Farbanteil und läßt diesen dann drucken (vorher müssen Sie natürlich auch das Farbband wechseln).

Die gefilterten Grafiken speichert man mit <S>. Mit <←> erscheint wieder die Orginalgrafikseite.

Zur Verbesserung der Bedienungsfeundlichkeit sind noch einige weitere nützliche Befehle vorhanden:

Durch die Taste <D> wird das Directory ausgegeben. Die Taste <C> dient zum Senden von Diskettenbefehlen. Mit der Taste <H> erreichen Sie die schon erwähnte Hilfstafel, und <Q> beendet das Programm. In Tabelle 1 finden Sie alle Befehle nochmals im Überblick.

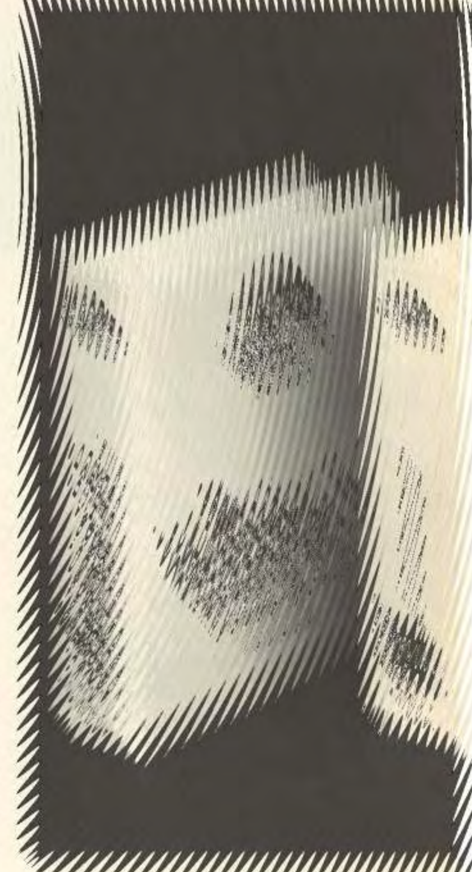
Und nun zu einem sehr wichtigen Teil des Programms, in den man mit <CMB P> gelangt: das Druckermenü. Hier kann das Programm an den eigenen Drucker angepaßt werden. Verändert werden können hier unter anderem die Sekundäradresse, der linke Rand, aber auch die Bildgröße. Außerdem kann hier eingestellt werden, ob der verwendete Drucker über die Möglichkeit verfügt, das Papier rückwärts zu transportieren. Angewählt werden die einzelnen Menüpunkte mit den angegebenen Tasten. Werte verändert man mit <+> und <->, mit <RETURN> werden sie bestätigt. Mit <Q> gelangt man wieder zur Grafikseite.

Falls sich Ihr Drucker dennoch nicht anpassen läßt, hängen Sie einfach Ihre eigene Druckroutine an. Sie müssen aber folgendes beachten:

1. Ihre Routine muß den Grafikbereich von $4000 bis $5FFF drucken. Falls die Grafikbytes um 90 Grad gedreht werden müssen, so verwendet man ein im Programm bereits vorhandenes Unterprogramm (Aufruf: JSR $0B40), welches zu Beginn und am Ende der Druckroutine aufgerufen werden muß. (Der erste Aufruf dient zum Drehen der Grafikbytes, der zweite zur Wiederherstellung der Grafik).

Bild 2. Ein zweites »Paint-Magic«-Bild als Hardcopy

2. Die Druckroutine muß ab der Adresse $0F91 nachgeladen werden, wobei die ersten 4 Bytes angeben müssen, ob die Druckroutine über ein Menü verfügt, in dem besondere Parameter eingestellt werden und ab welcher Adresse



dieses beginnt. Ist eine solche Routine vorhanden, so muß der Inhalt des ersten Bytes ($0F91) gleich 1 sein und in den Adressen $0F92, $0F93 und $0F94 der Aufruf des Menüs über einen JMP-Befehl erfolgen, wobei die Adresse $0F92 vom Hauptprogramm durch <CBM P> aufgerufen wird. Der Rücksprung aus dem Menü erfolgt mit JMP $09F1. Existiert kein Druckermenü, so muß der Inhalt von $0F91 gleich 0 gesetzt werden und die nächsten drei Bytes können beliebige Werte annehmen.

3. Die eigentliche Druckroutine beginnt bei $0F95 und wird mit RTS abgeschlossen.

Ihre eigene individuelle Version des Farbdruckprogramms erhalten Sie, wenn Sie nach dem Laden von »Obsess V3.1« Ihre angepaßte Druckroutine absolut nachladen und das Ganze mit einem Monitor von $0801 bis zum Ende der Druckroutine abspeichern.

Für den MPS 801/803 und Kompatible finden Sie in Listing 2 die bereits angepaßte Druckroutine »H-EXT« aus dem Programm »HARDMAKER« (64'er 9/1986). Sie muß nach der Eingabe mit dem MSE und dem Abspeichern lediglich wie oben beschrieben nachgeladen werden.

Alles gut und schön, werden jetzt viele sagen, aber wo bekomme ich die Farbbänder her? Bei einigen Farbbändern ist das kein Problem: in vielen gut bestückten Computergeschäften. Problematisch wird es allerdings oft mit gelben Farbbändern. Damit Sie auf keinen Fall Schwierigkeiten haben, finden Sie im Info einige Bezugsquellen.

(Gerhard Blickmann/Ralph Beninde/Ulrich Beenen)

Bezugsquellen für Farbbänder:

1. AGS-Farbbänder, Elektronik-Zubehör, Werwolf 54, 4650 Solingen 1, Tel. 02 12/1 30 84
   Die erhältlichen Farben sind rot, blau, grün und braun – beispielsweise für: MPS 801 16,50 Mark/Stück, MPS 802 17,80 Mark/Stück, MPS 803 13,60 Mark/Stück, Epson RX/FX 12,85 Mark/Stück, Star NL10 18,85 Mark/Stück, Seikosha SL/SP 16,90 Mark/Stück, Panasonic 15,80 Mark/Stück, (weitere Typen auf Anfrage)
2. Scanntronik, Parkstraße 38, 8011 Zorneding, Tel. 081 06/2 25 70
   Erhältlich ist ein Farbbandset, bestehend aus drei Bändern (rot, gelb, blau) für: Epson RX/FX 49 Mark/Stück, Star NL-10 59 Mark/Stück

Befehl	Funktion
<L>	Grafik laden
<S>	Grafik speichern
<D>	Directory zeigen
<C>	Diskettenbefehl senden
<←>	Zurück zur Originalgrafik
<1>-<4>	Filtern der Farbanteile
<F1>-<F7>	Farbwert ändern
<DEL>	Grafik verschieben <+> und <-> Schrittweite ändern (1-255)
<P>	Ausdrucken des gefilterten Bildes
<H>	Zeigen aller Befehle
<Q>	Verlassen des Programms
<CBM P>	Druckermenü

Tabelle 1. Alle Befehle auf einen Blick

```
Name : obsess v3.1            0801 1376
----------------------------------------
0801 : 0b 08 64 00 9e 32 30 36  d2
0809 : 31 00 00 00 a9 80 8d 8a  24
0811 : 02 20 b8 0a a2 00 bd 2b  0a
0819 : 08 c9 00 f0 07 20 d2 ff  e1
0821 : e8 4c 17 08 20 10 09 4c  35
0829 : f1 09 8e 08 1e 93 11 11  28
0831 : 11 11 11 11 12 20 20 20  14
0839 : 20 20 20 20 20 20 20 20  39
0841 : 20 20 20 20 20 20 20 20  41
0849 : 20 20 20 20 20 20 20 20  49
0851 : 20 20 20 20 20 20 20 20  51
0859 : 20 20 20 20 20 20 20 20  59
0861 : 20 20 20 20 20 20 20 20  61
0869 : 43 4f 4c 4f 52 50 52 49  d4
0871 : 4e 54 50 52 4f 47 52 41  43
0879 : 4d 20 20 20 20 20 20 20  a6
0881 : 20 20 20 20 20 20 20 20  81
0889 : 20 20 20 20 20 20 20 20  89
0891 : 20 20 20 20 20 20 20 20  91
0899 : 20 20 20 20 20 20 20 20  99
08a1 : 20 20 20 20 20 20 20 20  a1
08a9 : 20 20 20 20 20 11 11 11  d6
08b1 : 1f 20 28 43 29 20 31 39  1e
08b9 : 38 36 2f 38 37 20 42 59  0f
08c1 : 20 47 2e 42 4c 49 43 4b  0b
08c9 : 4d 41 4e 4e 20 26 20 52  6d
08d1 : 2e 42 45 4e 49 4e 44 45  de
08d9 : 20 92 00 a9 ff 8d 00 dd  a0
08e1 : 8d f0 09 a9 18 8d 16 d0  46
08e9 : a9 3b 8d 11 d0 a9 18 8d  8b
08f1 : 18 d0 60 a9 fe 8d 00 dd  d7
08f9 : 8d f0 09 a9 18 8d 16 d0  5e
0901 : a9 3b 8d 11 d0 a9 10 8d  83
0909 : 18 d0 60 00 00 00 00 a9  f5
0911 : 20 8d 1c 09 a9 00 8d 1b  27
0919 : 09 ad cc cc 8d 0f 09 a2  80
0921 : 04 a9 00 8d 0d 09 2e 0f  9c
0929 : 09 2e 0d 09 2e 0f 09 2e  8a
0931 : 0d 09 ad 0d 09 cd 0c 09  11
0939 : f0 0b 18 2e 0e 09 18 2e  61
0941 : 0e 09 4c 4e 09 38 2e 0e  d8
0949 : 09 38 2e 0e 09 ca d0 d1  8a
0951 : ad 1b 09 8d 64 09 ad 1c  fd
0959 : 09 18 69 20 8d 65 09 ad  50
0961 : 0e 09 8d cc cc ad 1b 09  a9
0969 : 18 69 01 8d 1b 09 ad 1c  11
0971 : 09 69 00 8d 1c 09 c9 40  92
0979 : f0 03 4c 1a 09 60 00 00  d5
0981 : 00 00 00 00 01 0b 0c 00  1a
0989 : 01 06 00 00 01 a2 05 ca  5c
0991 : bd 84 09 9d 7f 09 e0 00  4a
0999 : d0 f5 4c ae 09 a2 05 ca  9c
09a1 : bd 89 09 9d 7f 09 e0 00  dd
09a9 : d0 f5 4c ae 09 ad 7f 09  6b
09b1 : 8d 21 d0 a2 f9 ad 80 09  78
09b9 : 9d 00 d8 9d fa d8 [illegible] f4  17
09c1 : d9 9d ee da ad 81 09 18  bb
09c9 : 2a 18 2a 18 2a 18 2a 18  c9
09d1 : 6d 82 09 9d 00 04 9d fa  02
09d9 : 04 9d f4 05 9d ee 06 e0  b5
09e1 : 00 f0 04 ca 4c b6 09 ad  ae
09e9 : 83 09 8d 20 d0 60 00 00  68
09f1 : 20 dc 08 20 8e 09 a9 00  5d
09f9 : 85 c6 4c 01 0a 20 9e 09  43
0a01 : a5 c6 c9 00 f0 fa 20 b4  4c
0a09 : e5 c9 5f f0 46 c9 31 f0  22
0a11 : 45 c9 32 f0 41 c9 33 f0  f7
0a19 : 3d c9 34 f0 39 c9 50 f0  6b
0a21 : 3e c9 4c f0 46 c9 53 f0  57
0a29 : 45 c9 51 f0 63 c9 44 f0  3d
0a31 : 40 c9 43 f0 3f c9 48 f0  8a
0a39 : 3e c9 af f0 3d c9 14 f0  ba
0a41 : 46 c9 2b f0 45 c9 2d f0  8e
0a49 : 44 ae f0 09 e0 fe f0 b0  6d
0a51 : 4c 95 0a 4c dd 0a 4c e6  a1
0a59 : 0a 4c 08 0b 4c 24 0b 20  3f
0a61 : 46 0b 20 9b 0f 20 46 0b  c9
0a69 : 4c fe 09 4c 8f 0b 4c c6  10
0a71 : 0b 4c 59 0c 4c 32 0d 4c  9d
0a79 : fe 0e ad 97 0f c9 00 f0  fe
0a81 : 03 4c 98 0f 4c 01 0a 4c  40
0a89 : 3f 0f 4c 77 0f 4c 87 0f  e1
0a91 : 20 b8 0a 60 c9 85 f0 c1  ac
0a99 : c9 86 f0 bd c9 87 f0 b9  a9
0aa1 : c9 88 f0 b5 c9 89 f0 b4  b7
0aa9 : c9 8a f0 b0 c9 8b f0 ac  20
0ab1 : c9 8c f0 a8 4c 01 0a 20  47
0ab9 : 44 e5 a9 00 8d 20 d0 8d  92
0ac1 : 21 d0 a9 05 8d 86 02 a9  be
0ac9 : ff 8d 00 dd a9 1b 8d 11  16
0ad1 : d0 a9 15 8d 18 d0 a9 c8  ad
0ad9 : 8d 16 d0 60 20 8e 09 20  8c
0ae1 : dc 08 4c 01 0a a0 01 38  0f
0ae9 : e9 31 cd 0c 09 f0 13 8d  df
0af1 : 0c 09 20 9e 09 20 f4 08  d3
0af9 : c0 00 f0 03 20 10 09 4c  95
0b01 : 01 0a a0 00 4c f3 0a 38  2c
0b09 : e9 85 aa bd 7f 09 c9 0f  9d
0b11 : 30 08 a9 00 9d 84 09 4c  6a
0b19 : 40 0b 18 69 01 9d 84 09  33
0b21 : 4c 40 0b 38 e9 89 aa bd  68
0b29 : 84 09 c9 00 d0 08 a9 0f  b6
0b31 : 9d 84 09 4c 40 0b 38 e9  ed
0b39 : 01 9d 84 09 4c 40 0b 20  7e
0b41 : 8e 09 4c 01 0a a9 00 a0  b6
0b49 : 40 8d 5b 0b 8d 69 0b 8c  f1
0b51 : 5c 0b 8c 6a 0b a0 07 a2  ba
0b59 : 07 5e 08 40 6a ca 10 f9  cb
0b61 : 48 88 10 f3 a2 00 68 9d  77
0b69 : 08 40 e8 e0 08 d0 f7 18  ff
0b71 : ad 5b 0b 69 08 8d 5b 0b  2c
0b79 : 8d 69 0b ad 5c 0b 69 00  f7
0b81 : 8d 5c 0b 8d 6a 0b ad 6a  3b
0b89 : 0b c9 60 d0 c8 60 20 b8  2c
0b91 : 0a 20 03 0c a2 08 a0 00  db
0b99 : 20 ba ff ad 3c 03 a2 b5  9e
0ba1 : a0 0b 20 bd ff a2 00 a0  dd
0ba9 : 20 a9 00 20 d5 ff 20 14  a8
0bb1 : 0d 4c f1 09 20 20 20 20  46
0bb9 : 20 20 20 20 20 20 20 20  b9
0bc1 : 20 20 20 20 20 20 b8 0a  f7
0bc9 : 20 03 0c a2 08 a0 01 20  8c
0bd1 : ba ff ad 3c 03 a2 b5 a0  db
0bd9 : 0b 20 bd ff a2 00 a0 40  91
0be1 : 86 fb 84 fc a9 fb a2 00  2a
0be9 : a0 60 20 d8 ff 20 14 0d  48
0bf1 : 4c f1 09 1f 46 49 4c 45  c7
0bf9 : 4e 41 4d 45 4e 20 3a 20  f3
0c01 : 1e 00 a2 00 bd f4 0b c9  0b
0c09 : 00 f0 07 20 d2 ff e8 4c  b1
0c11 : 05 0c a2 00 20 cf ff 9d  81
0c19 : b5 0b e8 e0 11 f0 04 c9  e6
0c21 : 0d d0 f1 8e 3c 03 a9 01  69
0c29 : a2 08 a0 0f 20 ba ff a9  04
0c31 : 02 a2 42 a0 0c 20 bd ff  e2
0c39 : 20 c0 ff a9 01 20 c3 ff  0e
0c41 : 60 55 3a 20 20 20 20 20  a2
0c49 : 20 20 20 20 20 20 20 20  49
0c51 : 20 20 20 24 00 00 00 00  0e
0c59 : 20 b8 0a 20 27 0c a2 08  c9
0c61 : a0 00 8c 86 02 8c 56 0c  eb
0c69 : 8c 55 0c 20 ba ff a9 01  fb
0c71 : a2 54 a0 0c 20 bd ff a9  2a
0c79 : 00 8d 55 0c a2 00 a0 60  84
0c81 : 20 d5 ff 8e 57 0c 8c 58  16
0c89 : 0c a9 05 8d 86 02 a9 93  a3
0c91 : 20 d2 ff a2 00 8e 9f 0c  7a
0c99 : a2 60 8e a0 0c ad cc cc  1e
0ca1 : c9 22 f0 38 ae 55 0c e0  46
0ca9 : 01 d0 03 20 d2 ff 18 ad  c0
0cb1 : 9f 0c 69 01 8d 9f 0c ad  32
0cb9 : a0 0c 69 00 8d a0 0c cd  63
0cc1 : 58 0c d0 d9 ad 9f 0c cd  32
0cc9 : 57 0c d0 d1 a9 00 8d 57  14
0cd1 : 0c a9 60 8d 58 0c 20 14  0a
0cd9 : 0d 4c f1 09 ae 55 0c e0  31
0ce1 : 00 d0 08 a2 01 8e 55 0c  92
```

Listing 1. Das Farbdruckprogramm »Obsess V3.1«, bitte mit dem MSE eingeben

```
0ce9 : 4c af 0c a9 00 8d 55 0c  1f
0cf1 : a9 0d 20 d2 ff ee 56 0c  6c
0cf9 : ae 56 0c e0 18 f0 09 4c  b7
0d01 : af 0c 20 14 0d 4c f1 09  4e
0d09 : a2 00 8e 56 0c 20 14 0d  46
0d11 : 4c 00 0d a9 00 85 c6 a5  68
0d19 : c6 c9 00 f0 fa 60 1f 44  9a
0d21 : 49 53 4b 2d 4b 4f 4d 4d  8b
0d29 : 41 4e 44 4f 20 3a 20 1e  1d
0d31 : 00 20 b8 0a a2 00 bd 1f  10
0d39 : 0d c9 00 f0 07 20 d2 ff  06
0d41 : e8 4c 37 0d 20 13 0c a9  dd
0d49 : 01 a2 08 a0 0f 20 ba ff  8e
0d51 : ad 3c 03 a2 b5 a0 0b 20  fe
0d59 : bd ff 20 c0 ff a9 01 20  c7
0d61 : c3 ff 20 14 0d 4c f1 09  bc
0d69 : 93 12 1e 20 42 45 46 45  83
0d71 : 48 4c 53 4c 49 53 54 45  49
0d79 : 20 92 11 11 0d 20 20 20  db
0d81 : 3c 4c 3e 20 20 20 20 4d  95
0d89 : 55 4c 54 49 43 4f 4c 4f  c1
0d91 : 52 42 49 4c 44 20 4c 41  d9
0d99 : 44 45 4e 0d 20 20 20 3c  b1
0da1 : 53 3e 20 20 20 20 47 45  ca
0da9 : 46 49 4c 54 45 52 54 45  f4
0db1 : 53 20 42 49 4c 44 20 53  dc
0db9 : 50 45 49 43 48 45 52 4e  fb
0dc1 : 0d 20 20 20 3c 44 3e 20  0a
0dc9 : 20 20 20 44 49 52 45 43  4d
0dd1 : 54 4f 52 59 0d 20 20 20  1f
0dd9 : 3c 43 3e 20 20 20 20 44  57
0de1 : 49 53 4b 2d 4b 4f 4d 4d  4b
0de9 : 41 4e 44 4f 20 53 45 4e  9b
0df1 : 44 45 4e 11 0d 20 20 20  20
0df9 : 3c 5f 3e 20 20 20 20 4f  9b
0e01 : 52 49 47 49 4e 41 4c 47  a2
0e09 : 52 41 46 49 4b 20 5a 45  60
0e11 : 49 47 45 4e 0d 20 20 3c  e4
0e19 : 31 2d 34 3e 20 20 20 46  c6
0e21 : 41 52 42 45 20 49 53 4f  fd
0e29 : 4c 49 45 52 45 4e 20 26  49
0e31 : 20 42 49 4c 44 20 5a 45  87
0e39 : 49 47 45 4e 0d 20 3c 46  90
0e41 : 31 2d 46 38 3e 20 20 46  93
0e49 : 41 52 42 45 4e 20 41 45  62
0e51 : 4e 44 45 52 4e 11 0d 20  3f
0e59 : 20 3c 44 45 4c 3e 20 20  c9
0e61 : 20 4f 52 49 47 49 4e 41  61
0e69 : 4c 47 52 41 46 49 4b 20  32
0e71 : 56 45 52 53 43 48 49 45  8f
0e79 : 42 45 4e 0d 20 20 3c 2b  dd
0e81 : 2f 2d 3e 20 20 20 56 45  c1
0e89 : 52 53 43 48 49 45 42 45  b1
0e91 : 53 43 48 52 49 54 54 57  19
0e99 : 45 49 54 45 11 0d 20 20  7b
0ea1 : 20 3c 50 3e 20 20 20 20  7f
0ea9 : 49 53 4f 4c 49 45 52 54  aa
0eb1 : 45 20 46 41 52 42 45 20  4d
0eb9 : 44 52 55 43 4b 45 4e 0d  16
0ec1 : 20 20 3c 43 3d 50 3e 20  f8
0ec9 : 20 20 44 52 55 43 4b 50  92
0ed1 : 41 52 41 4d 45 54 45 52  e6
0ed9 : 20 41 45 4e 44 45 52 4e  09
0ee1 : 11 0d 20 20 20 3c 51 3e  2b
0ee9 : 20 20 20 20 50 52 4f 47  89
0ef1 : 52 41 4d 4d 20 42 45 45  95
0ef9 : 4e 44 45 4e 00 20 b8 0a  7c
0f01 : a9 69 8d 0c 0f a9 0d 8d  d1
0f09 : 0d 0f ad cc cc c9 00 f0  a0
0f11 : 17 20 d2 ff 18 ad 0c 0f  2a
0f19 : 69 01 8d 0c 0f ad 0d 0f  98
0f21 : 69 00 8d 0d 0f 4c 0b 0f  2d
0f29 : 18 a2 0d a0 22 20 f0 ff  d1
0f31 : a9 00 ae 50 0f 20 cd bd  35
0f39 : 20 14 0d 4c f1 09 a9 00  3e
0f41 : 8d 52 0f 8d 57 0f a9 20  42
0f49 : 8d 53 0f 8d 58 0f a2 01  80
0f51 : bd cc cc a2 00 9d cc cc  b5
0f59 : 18 ad 52 0f 69 01 8d 52  38
0f61 : 0f 8d 57 0f ad 53 0f 69  73
0f69 : 00 8d 53 0f 8d 58 0f c9  52
0f71 : 48 d0 db 4c f1 09 ae 50  64
0f79 : 0f e0 ff f0 06 ee 50 0f  4d
0f81 : ee 20 d0 4c 01 0a ae 50  f9
0f89 : 0f e0 01 f0 f6 ce 50 0f  ac
0f91 : ee 20 d0 4c 01 0a 01 4c  4a
0f99 : 52 11 a2 00 86 f7 a2 40  50
0fa1 : 86 f8 a9 01 a2 04 a0 00  fb
0fa9 : 20 ba ff a9 00 20 bd ff  53
0fb1 : 20 c0 ff a2 01 20 c9 ff  be
0fb9 : a9 1b 20 d2 ff a9 41 20  e5
0fc1 : d2 ff a9 08 20 d2 ff a0  d8
0fc9 : 00 8c 00 80 a2 14 a9 20  d1
0fd1 : e0 00 f0 07 20 d2 ff ca  fd
0fd9 : 4c d1 0f a9 1b 20 d2 ff  05
0fe1 : a9 2a 20 d2 ff a9 00 20  8f
0fe9 : d2 ff a9 40 20 d2 ff a9  19
0ff1 : 01 20 d2 ff a0 00 8c 01  f5
0ff9 : 80 8c 02 80 a2 00 8e 01  b6
1001 : 80 a2 00 8e 02 80 b1 f7  7f
1009 : 20 d2 ff 18 a5 f7 69 01  57
1011 : 85 f7 a5 f8 69 00 85 f8  b9
1019 : ee 02 80 ee 20 d0 ae 02  4d
1021 : 80 e0 28 d0 e1 ee 01 80  d0
1029 : ee 21 d0 ad 01 80 c9 08  dd
1031 : d0 cf a9 0a 20 d2 ff a9  80
1039 : 0d 20 d2 ff a2 19 ee 00  ba
1041 : 80 ae 00 80 e0 19 d0 84  4c
1049 : 4c 5e 10 a9 1b 20 d2 ff  fc
1051 : a9 6a 20 d2 ff a9 18 20  80
1059 : d2 ff ca d0 ee 20 cc ff  1b
1061 : a9 01 20 c3 ff 60 12 1e  93
1069 : 20 20 44 52 55 43 4b 50  32
1071 : 41 52 41 4d 45 54 45 52  86
1079 : 20 41 45 4e 44 45 52 4e  a9
1081 : 20 92 11 11 0d 20 20 20  e3
1089 : 3c 31 3e 20 20 4e 55 52  60
1091 : 20 43 52 11 0d 20 20 20  9c
1099 : 3c 32 3e 20 20 4e 55 52  f0
10a1 : 20 4c 46 11 0d 20 20 20  2e
10a9 : 3c 33 3e 20 20 4c 46 20  d0
10b1 : 55 4e 44 20 43 52 11 11  70
10b9 : 0d 20 20 20 3c 34 3e 20  81
10c1 : 20 50 41 50 49 45 52 52  10
10c9 : 55 45 43 4b 4c 41 55 46  ac
10d1 : 20 4d 4f 45 47 4c 49 43  97
10d9 : 48 11 0d 20 20 20 3c 35  50
10e1 : 3e 20 20 50 41 50 49 45  88
10e9 : 52 52 55 45 43 4b 4c 41  a5
10f1 : 55 46 20 4e 49 43 48 54  b4
10f9 : 20 4d 4f 45 47 4c 49 43  bf
1101 : 48 11 11 0d 20 20 20 3c  b4
1109 : 51 3e 20 20 51 55 49 54  13
1111 : 11 11 0d 20 20 20 3c 54  8f
1119 : 3e 20 20 4c 49 4e 4b 45  b8
1121 : 52 20 52 41 4e 44 11 0d  a5
1129 : 20 20 20 3c 53 3e 20 20  d1
1131 : 53 45 4b 2d 41 44 52 45  a9
1139 : 53 53 45 11 0d 20 20 20  3c
1141 : 3c 42 3e 20 20 42 49 4c  04
1149 : 44 47 52 4f 45 53 53 45  76
1151 : 00 20 b8 0a a2 00 bd 67  c1
1159 : 10 c9 00 f0 07 20 d2 ff  29
1161 : e8 4c 57 11 20 28 12 8d  0e
1169 : 20 d0 20 0d 13 a5 c6 c9  a8
1171 : 00 f0 f4 20 b4 e5 c9 31  2e
1179 : f0 23 c9 32 f0 28 c9 33  91
1181 : f0 2d c9 34 f0 32 c9 35  33
1189 : f0 34 c9 51 f0 36 c9 54  c0
1191 : f0 38 c9 53 f0 37 c9 42  ef
1199 : f0 36 4c 68 11 20 d4 11  4c
11a1 : 20 00 12 4c 68 11 20 f0  41
11a9 : 11 20 e4 11 4c 68 11 20  b2
11b1 : f0 11 20 d4 11 4c 68 11  04
11b9 : 20 1c 12 4c 68 11 20 0c  9d
11c1 : 12 4c 68 11 20 f4 08 4c  98
11c9 : fe 09 4c 6b 12 4c 70 12  36
11d1 : 4c 75 12 a9 20 8d 3a 10  09
11d9 : a9 d2 8d 3b 10 a9 ff 8d  1f
11e1 : 3c 10 60 a9 ea 8d 3a 10  97
11e9 : 8d 3b 10 8d 3c 10 60 a9  e3
11f1 : 20 8d 35 10 a9 d2 8d 36  fb
11f9 : 10 a9 ff 8d 37 10 60 a9  58
1201 : ea 8d 35 10 8d 36 10 8d  e7
1209 : 37 10 60 a9 20 8d 49 10  49
1211 : a9 5e 8d 4a 10 a9 10 8d  40
1219 : 4b 10 60 a9 ea 8d 49 10  1a
1221 : 8d 4a 10 8d 4b 10 60 a9  93
1229 : 05 8d 86 02 18 a2 12 a0  f7
1231 : 16 20 f0 ff a9 00 ae ce  86
1239 : 0f 20 cd bd a9 20 20 d2  45
1241 : ff 18 a2 14 a0 16 20 f0  95
1249 : ff a9 00 ae a8 0f 20 cd  12
1251 : bd a9 20 20 d2 ff 20 d2  42
1259 : ff 18 a2 16 a0 16 20 f0  ed
1261 : ff a9 00 ae e7 0f 20 cd  1e
1269 : bd 60 a2 54 4c 7a 12 a2  b0
1271 : 53 4c 7a 12 a2 42 4c 7a  2e
1279 : 12 8e a1 12 8e af 12 a9  7f
1281 : 06 8d 20 d0 20 28 12 a5  47
1289 : c6 c9 00 f0 fa 20 b4 e5  a1
1291 : c9 2b f0 0b c9 2d f0 15  81
1299 : c9 0d f0 1f 4c 88 12 a9  ad
12a1 : 00 c9 54 f0 19 c9 53 f0  c8
12a9 : 22 c9 42 f0 2b a9 00 c9  f2
12b1 : 54 f0 32 c9 53 f0 3b c9  80
12b9 : 42 f0 44 4c 68 11 ad ce  71
12c1 : 0f c9 46 f0 03 ee ce 0f  65
12c9 : 4c 85 12 ad a8 0f c9 ff  3c
12d1 : f0 03 ee a8 0f 4c 85 12  a1
12d9 : ad e7 0f c9 07 f0 03 ee  59
12e1 : e7 0f 4c 85 12 ad ce 0f  fb
12e9 : c9 00 f0 03 ce ce 0f 4c  87
12f1 : 85 12 ad a8 0f c9 00 f0  21
12f9 : 03 ce a8 0f 4c 85 12 ad  04
1301 : e7 0f c9 00 f0 03 ce e7  14
1309 : 0f 4c 85 12 ad 35 10 a2  ec
1311 : 05 8e 78 d8 8e c8 d8 8e  46
1319 : 90 d9 8e e0 d9 c9 ea d0  8f
1321 : 10 a9 3e 8d 78 04 a9 20  d6
1329 : 8d c8 04 8d 18 05 4c 59  5b
1331 : 13 ad 3a 10 c9 ea d0 10  03
1339 : a9 3e 8d c8 04 a9 20 8d  a7
1341 : 78 04 8d 18 05 4c 59 13  60
1349 : a9 3e 8d 18 05 a9 20 8d  b1
1351 : 78 04 8d c8 04 4c 59 13  76
1359 : ad 49 10 c9 ea f0 0b a9  9e
1361 : 3e 8d e0 05 a9 20 8d 90  32
1369 : 05 60 a9 3e 8d 90 05 a9  95
1371 : 20 8d e0 05 60 bd 20 1e  e1
```

Listing 1. (Schluß)

```
Name : mps 801                0f91 118c
----------------------------------------
0f91 : 00 00 00 00 a9 40 8d ea  3a
0f99 : 10 a9 04 85 ba a2 00 86  fd
0fa1 : 90 86 fe 20 b1 ff 20 ae  31
0fa9 : ff a6 90 f0 01 60 86 b9  de
0fb1 : 86 b7 e8 86 b8 20 c0 ff  ad
0fb9 : a6 b8 20 c9 ff a9 ff 85  55
0fc1 : 61 a9 07 8d eb 10 a9 1c  88
0fc9 : 85 97 a9 00 8d e6 10 a9  28
0fd1 : 28 8d e8 10 a2 04 bd c3  c5
0fd9 : 10 20 f5 10 ca 10 f7 a9  d9
0fe1 : 00 85 63 85 64 ad e6 10  9d
0fe9 : 85 65 a9 00 8d ec 10 a5  57
0ff1 : 63 a6 64 a4 65 20 88 10  ef
0ff9 : ae ec 10 a5 ad a0 00 b1  19
1001 : ac ae ec 10 9d ed 10 e6  99
1009 : 65 e8 8e ec 10 ec eb 10  5c
1011 : d0 dd a9 00 a0 07 d0 02  c4
1019 : d0 b5 ae eb 10 1e ed 10  b7
1021 : 2a ca 10 f9 25 61 09 80  76
1029 : 20 f5 10 ad 8d 02 29 01  8d
1031 : d0 f9 a5 91 10 3d 88 10  c7
1039 : e1 a5 63 18 69 08 85 63  7c
1041 : 90 02 e6 64 ce e8 10 d0  2f
1049 : 9c a9 0d 20 f5 10 ad e6  66
1051 : 10 18 69 07 8d e6 10 c6  86
1059 : 97 f0 02 d0 bb a9 04 cd  b8
1061 : eb 10 f0 0f 8d eb 10 a9  3e
1069 : 01 85 97 a9 0f 85 61 d0  8c
1071 : ea a9 01 85 fe a9 0f 20  db
1079 : f5 10 a9 0d 20 f5 10 20  b5
1081 : cc ff a9 01 4c c3 ff 85  c5
1089 : 14 86 15 98 4a 4a 4a aa  ae
1091 : bd c8 10 85 ad 8a 29 03  41
1099 : aa bd e2 10 85 ac 98 29  4f
10a1 : 07 18 65 ac 85 ac a5 14  20
10a9 : 29 f8 85 63 ad ea 10 18  bf
10b1 : 65 ad 85 ad a5 ac 18 65  ef
10b9 : 63 85 ac a5 ad 65 15 85  24
10c1 : ad 60 50 00 10 1b 08 00  ac
10c9 : 01 02 03 05 06 07 08 0a  fa
10d1 : 0b 0c 0d 0f 10 11 12 14  02
10d9 : 15 16 17 19 1a 1b 1c 1e  0a
10e1 : 1f 00 40 80 c0 00 00 00  2d
10e9 : 00 00 00 00 00 00 00 00  ea
10f1 : 00 00 00 00 8e 87 11 ae  b8
10f9 : 88 11 d0 16 c9 08 d0 0b  37
1101 : a2 00 8e 89 11 8e 8a 11  4a
1109 : 8d 88 11 20 dd ed ae 87  3a
1111 : 11 60 c9 0f d0 0c 20 52  39
1119 : 11 a9 00 8d 88 11 a9 0f  86
1121 : d0 e9 aa 30 10 48 20 52  ff
1129 : 11 a9 00 8d 89 11 8d 8a  2d
1131 : 11 68 4c 0c 11 cd 89 11  d3
1139 : d0 08 ee 8a 11 d0 cf ce  8f
1141 : 8a 11 48 20 52 11 68 8d  d4
1149 : 89 11 a9 01 8d 8a 11 d0  f8
1151 : bd c9 0d d0 09 ad 89 11  96
1159 : c9 80 d0 02 f0 b0 ae 8a  3b
1161 : 11 f0 ab e0 03 b0 0b ad  2e
1169 : 89 11 20 dd ed ca d0 f7  a7
1171 : f0 9c a9 1a 20 dd ed ad  61
1179 : 8a 11 20 dd ed ad 89 11  e4
1181 : 20 dd ed 4c 0f 11 00 00  0e
1189 : 08 01 c9 03 b2 ee 43 02  98
```

Listing 2. Die Druckroutine für MPS 801 und Kompatible, bitte mit dem MSE (Seite 159) eingeben

Magie im Bildschirmrahmen

Lange Zeit glaubte man, der Bildschirmrahmen des C64 sei bis in alle Ewigkeit zur Nutzlosigkeit verdammt. Doch »Magic Border Beams« beweist das Gegenteil. Mit einer ausgefeilten Interrupt-Technik werden faszinierende Effekte im Bildschirmrahmen erzeugt.

Das Programm-Paket mit dem Namen »Magic Border Beams« erlaubt den Entwurf von eindrucksvollen bewegten Rahmen-Mustern mit einem komfortablen Editor und deren Verwendung in Basic-Programmen (Bild 1). Da pro Rasterzeile eine Farbe erlaubt ist, kann man leicht fließend wirkende Farbübergänge und metallisch wirkende Effekte erzeugen. Weil das Foto nur einen Schnappschuß eines einzigen Bildes darstellt, läßt sich der faszinierende Effekt leider nicht ganz vermitteln.

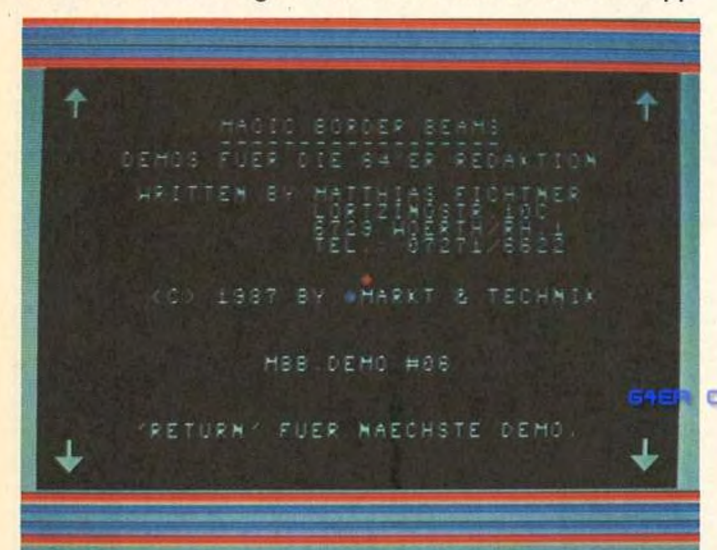


Bild 1. Das Programm »Magic Border Beams« in Aktion

Das Herzstück von »Magic Border Beams« (MBB) ist eine Rasterinterrupt-Routine, die es ermöglicht, absolut flimmerfreie »Trickfilme« im Bildschirmrahmen des C64 ober- und unterhalb des Bildschirmfensters einzublenden. Ein solcher MBB-Film besteht aus bis zu 256 Einzelbildern, die jeweils 24 Rasterzeilen hoch sind und die gesamte Bildschirmbreite ausnutzen. Eine ausgefeilte IRQ-Technik erlaubt es, jeder der 24 Rasterzeilen eine eigene Farbe zu geben. Durch geeignete Veränderung der Farbkombinationen eines jeden Bildes lassen sich verblüffende Animationseffekte erzielen.

Eigene Filme erzeugen

Es versteht sich von selbst, daß MBB-Filme unabhängig von anderen Basic- und Maschinenprogrammen im Interrupt des Commodore 64, und somit in einer Art Pseudo-Multitasking ablaufen. Man muß hier jedoch einen Geschwindigkeitsverlust des Hauptprogramms von rund 17 Prozent in Kauf nehmen. Des weiteren kann die systeminterne Variable TI$ nicht mehr verwendet werden.

Beim Erstellen von MBB-Filmen hilft ein komfortabler Editor, der in Basic geschrieben ist (Listing 1). Nach dem Abtippen mit dem Checksummer sollten Sie diesen unter dem Namen »MBB-EDI.SCR« auf Diskette speichern.

Zeit- und arbeitsintensive Aufgaben übernehmen verschiedene Assembler-Routinen, die in der Datei »MBB-EDI.ASS« enthalten sind. Das dazugehörige MSE-Listing finden Sie in Listing 2.

Um eine komfortable Steuerung der MBB-Filme zu ermöglichen, bietet »Magic Border Beams« zusätzlich eine Basic-Erweiterung, die zusammen mit den wichtigen Routinen zur Behandlung des Rasterzeilen-Interrupts in der Datei »MBB-IRQ.CDE« enthalten ist. Listing 3 zeigt das entsprechende MSE-Listing. Es wird mit einem kleinen Basic-Programm initialisiert und aktiviert (Listing 4). Für die Eingabe verwenden Sie auch hier bitte den Checksummer. Anschließend muß es unter dem Namen »MBB-IRQ.BOT« gespeichert werden.

Rasterzeilen-Editor der Spitzenklasse

Nach Eingabe aller Programmteile sollten sich folgende Dateien auf der Diskette befinden:

1. MBB-EDI.SCR
2. MBB-EDI.ASS
3. MBB-IRQ.BOT
4. MBB-IRQ.CDE

Damit ist MBB vollständig und kann aktiviert werden. Der Editor wird nun mit

```
LOAD "MBB-EDI.SCR",8
```

geladen. Startet man das Programm mit RUN, werden zunächst zwei Files nachgeladen (»MBB-EDI.ASS« und »MBB-IRQ.CDE«). Nach einem einführenden Titelbild gelangt man mit <RETURN> in den Editor.

Hier wird jeweils ein Bild des MBB-Films in achtfacher Vergrößerung auf dem Bildschirm dargestellt. Die oberste Zeile des Bildschirms steht für die Kommunikation mit dem Programm zur Verfügung. Am linken Bildschirmrand sieht man einen Pfeil (siehe Bild 2), der die gerade bearbeitete Zeile anzeigt. Er läßt sich mit den Tasten <CRSR aufwärts/abwärts> bewegen.

Bei der Anwahl einer Funktion erhalten Zahlen, die in der Kommunikationszeile einzugeben sind, einen voreingestellten Wert und können mit Hilfe der Tasten <+>, <-> in Einer- und mit <SHIFT +> und <SHIFT -> in Zehnerschritten erhöht beziehungsweise vermindert werden. Sie sind anschließend mit <RETURN> zu bestätigen. Handelt es sich bei der Zahl um die Nummer eines Bildes, wird während der Einstellung immer das entsprechende Bild eingeblendet.

Gleiches gilt auch, wenn ein Farbwert verlangt wird. Hierbei wird die gewählte Farbe in einem kleinen Kästchen in der Kommunikationszeile dargestellt. Man erspart sich somit das umständliche Hantieren mit den Farbcodes.

Verlangt der Computer eine Ja/Nein-Entscheidung, so antwortet man mit den Tasten <Y> oder <N>, und bestätigt anschließend mit <RETURN>.

Bei fehlerhaften Eingaben kann man die Ausführung fast aller Befehle verhindern, indem man so lange <RETURN> drückt, bis man sich wieder im Ausgangsmodus befindet.

Zur Bearbeitung eines Films stehen auf Tastendruck folgende Befehle oder Tasten zur Steuerung zur Verfügung:

Cursor-Tasten (<CRSR aufwärts>, <CRSR abwärts>)
Der Cursor wird um eine Zeile nach oben beziehungs-

Sonderheft 12: Eine wahre Fundgrube für Programmierfreaks!

In dieser Sonderheft-Ausgabe von 64'er erfahren Assembler-Programmierer alles über die Programmierung von Pull Down Menues. Zusätzlich bietet der Maschinensprache-Monitor »Promon« hochwertiges Assembler-Werkzeug. Ein ausführlicher Pascal-Kurs stellt u.a. die Besonderheiten von Pascal für den C64 und von Turbo-Pascal für den C128 heraus. Wer sich für andere Sprachen interessiert, findet Berichte, Vorstellungen und viele Praxis-Tips zu Comnal, C, Forth und der KI-Sprache Prolog.

Nutzen Sie die Bestellmöglichkeit des zwölften 64'er-Sonderheftes »C64-Grundwissen« mit der eingehefteten Zahlkarte in diesem Sonderheft von »64'er«!

weise unten bewegt und zeigt auf die zu bearbeitende Rasterzeile.

Farb-Tasten (<CTRL 1-8>, <CBM 1-8>)
Die Farbe der aktuellen Zeile wird entsprechend dem Farbcode der betätigten Tastenkombination verändert.

<+> <->, <SHIFT +> <SHIFT ->
Durch diese Tasten wählt man das nächste (<+>) oder vorhergehende Bild (<->) des Filmes an. In Verbindung mit <SHIFT> geschieht dies in Zehnerschritten. Mit <F1> und <F3> springt man stets auf das Bild Nummer 0, mit <F5> und <F7> auf Bild Nummer 128.

<L> (Load)
Dieser Befehl lädt einen Film von der Diskette. Hierzu wird in der Kommunikationszeile nach dem Namen des zu ladenden Films gefragt. Dabei wird die Kennung »MBB.« für Magic-Border-Filme vorgegeben. Drückt man sofort <RETURN>, wird der Befehl ignoriert. Ansonsten wird der entsprechende Film geladen und entpackt (mehr dazu unter »Save«).

<S> (Save)
Speichert den aktuellen Film auf Diskette. Zur Eingabe des Filenamens gilt das bei »L« (Load) Gesagte. Gibt man einen Namen ein, wird der Film gepackt und gespeichert. Durch das Packen wird die Länge des Films von 8 KByte auf 4 KByte verkürzt.

<P> (Parameter)
Dieser Befehl legt die spezifischen Parameter des Films fest.
In der Kommunikationszeile wird man zunächst nach der Ablaufgeschwindigkeit des Films gefragt. Diese liegt zwischen 0 (schnell) und 31 (langsam). Nun verlangt das Programm die Nummern des ersten und des letzten Bildes, die zum Film gehören (Näheres zu diesen beiden Zahlen siehe unter »!M« bei der Basic-Erweiterung).

Komfortable Befehle

Anschließend erhält man die Möglichkeit, eine bestimmte Farbe als »transparent« zu definieren. Dies bedeutet, daß an allen Stellen des Films, an denen diese »transparente« Farbe auftaucht, die Farbe des Bildschirmrahmens eingeblendet wird. Beantwortet man die Frage mit »Y«, so ist die transparent wirkende Farbe mit <+> oder <-> anzugeben. Diese Parameter werden beim Speichern zusätzlich auf Diskette abgelegt und beim Laden (auch von der Basic-Erweiterung aus) neu gesetzt.

<C> (Change Color)
Dieser Befehl dient dazu, eine bestimmte Farbe in einem bestimmten Abschnitt des Films durch eine andere zu ersetzen. Zunächst verlangt das Programm die Eingabe der alten und neuen Farbe (mit <+> und <->). Danach muß man die Nummern des ersten und letzten Bildes eingeben, in denen die Farbe geändert werden soll.

<T> (Transfer Pictures)
Mit diesem Befehl kann man einen beliebigen Ausschnitt des Films an eine andere Stelle kopieren. Dazu benötigt das Programm die Nummern des ersten und letzten zu kopierenden Bildes, sowie die Bildnummer, ab der der kopierte Ausschnitt erscheinen soll.

<F> (Transfer Picture from File)
Dieser Befehl ist dem T-Befehl sehr ähnlich, erlaubt jedoch das Kopieren von Ausschnitten eines auf Diskette gespeicherten Films. Hierzu gibt man zunächst den Namen des Quellfiles ein, wobei wieder »MBB.« vorgegeben wird. Drückt man nur <RETURN>, wird die Ausführung abgebrochen. Andernfalls verlangt der Computer die Nummern des ersten und letzten zu kopierenden Bildes des Quellfilms. Nach Eingabe der Nummer des Zielbildes wird der gewählte Ausschnitt des gespeicherten Films in den gerade bearbeiteten Film eingebunden.

<I> (Invert Order)
Dieser Befehl dient dazu, die Reihenfolge der Bilder in einem bestimmten Abschnitt des Films zu invertieren. Dadurch ist es möglich, Teile des Films (oder den ganzen Film) rückwärts ablaufen zu lassen. Hierfür ist die Eingabe der Nummern des ersten und letzten Bildes des zu invertierenden Abschnitts nötig.

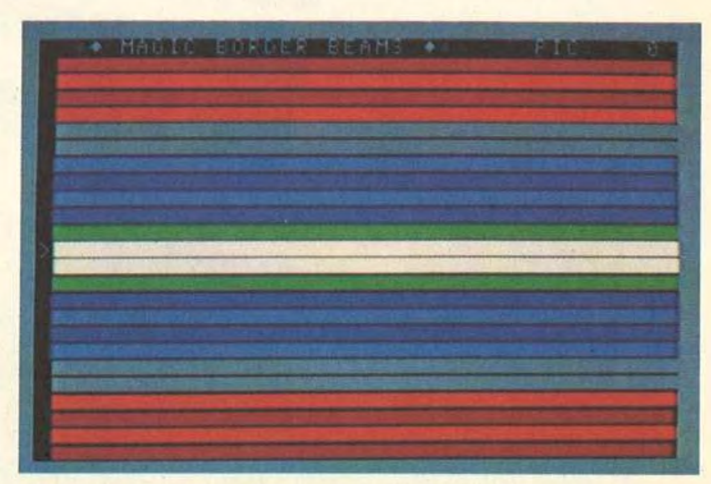


Bild 2. Der Editor von »Magic Border Beams« bei der Bearbeitung einer Filmsequenz. Jedes einzelne Bild kann verändert werden.

<B> (Border)
Die Rahmenfarbe des Bildschirms wird bei jedem Betätigen von <B> verändert.

<U> (Up)
Das aktuelle Bild wird um eine Zeile nach oben gescrollt.

<D> (Down)
Das aktuelle Bild wird um eine Zeile nach unten gescrollt.

<SHIFT CLR/HOME> (Clear)
Das aktuelle Bild wird gelöscht und mit der Rahmenfarbe gefüllt.

<=> (Bild kopieren)
Das vorhergehende Bild wird in das aktuelle kopiert. Diese Funktion ist sehr nützlich, wenn sich das aktuelle Bild aus animationstechnischen Gründen nur minimal vom vorhergehenden unterscheiden soll und nur kleine Veränderungen vorgenommen werden.

<M> (Zwischenspeichern)
Mit diesem Befehl kann man einen beliebigen Ausschnitt des aktuellen Bildes zwischenspeichern. Zunächst wählt man einen der acht zur Verfügung stehenden Speicher aus (an dieser Stelle kann man die Ausführung des Befehls durch <←> abbrechen). Anschließend bestimmt man mit dem Cursor die unterste Zeile des Abschnittes, die gespeichert werden soll, und bestätigt mit <RETURN>. Nun kann man den Cursor oberhalb dieser Zeile bewegen, um die oberste Zeile des Ausschnittes festzulegen. Auch hier wird mit <RETURN> bestätigt.

<R> (Recall)
Mit diesem Befehl kann man einen mit dem M-Befehl zwischengespeicherten Bildausschnitt in das aktuelle Bild

einbinden, wobei die aktuelle Position des Cursors den unteren Rand der Einbindung festlegt. In der Kommunikationszeile muß man lediglich die Nummer des zu benutzenden Speichers (siehe <M>) angeben. Auch hier führt die <←>-Taste zum Abbruch.

<N> (New)
Der komplette Film wird nach einer Sicherheitsabfrage gelöscht und mit der aktuellen Rahmenfarbe gefüllt.

<G> (Go)
Der Film wird entsprechend den mit <P> (Parameter) gesetzten Werten im Bildschirmrahmen gezeigt. In diesem Modus kann man mit <B> die Rahmenfarbe verändern. Mit <H> (Halt) gelangt man wieder in den Normalmodus.

<Q> (Quit)
Nach einer Sicherheitsabfrage wird der Editor durch einen Reset verlassen.

Diese Befehle mögen teilweise recht komplex erscheinen, sind jedoch sehr einfach und komfortabel zu bedienen, so daß man schon nach kurzer Zeit herrliche MBB-Filmsequenzen entwerfen kann.

Wer seine so erstellten Werke in Basic-Programmen verwenden will, der kann die folgende MBB-Basic-Erweiterung zu Hilfe nehmen.

Bevor man die Basic-Erweiterung lädt, sollte man alle eventuellen Erweiterungen (zum Beispiel Formel 64) abschalten, da sich sonst Komplikationen ergeben könnten.

Nach dem Starten des Ladeprogramms »MBB-IRQ.BOT« mit RUN wird zunächst das Maschinenprogramm »MBB-IRQ.CDE« nachgeladen. Es erscheint ein Titelbild. Mit <RETURN> gelangt man ins eigentliche Programm. Von nun an stehen folgende neue Befehle sowohl im Direkt-, als auch im Programm-Modus zur Verfügung:

!L,"MBB.Film" (Load)
Dieser Befehl lädt einen mit dem Editor erstellten Film mit dem Namen »Film«, entpackt ihn und setzt die im Editor definierten Filmparameter (Geschwindigkeit, erstes/letztes Bild, »transparente« Farbe). Bis auf die transparente Farbe können alle Parameter durch die Befehle »!S« (Speed) und »!M« (Movie) geändert werden.

!S,x (Speed)
legt die Geschwindigkeit fest, mit der der Film gezeigt wird. Der Parameter »x« liegt zwischen 0 und 32, wobei 0 »Zeitraffer« und 32 »Standbild« bedeutet.

!M,x,y (Movie)
definiert eine Filmsequenz, wobei »x« das erste und »y« das letzte zu zeigende Bild ist. x und y liegen zwischen 0 und 255. Sollte x größer als y sein, so werden die Bilder x bis 255 und anschließend 0 bis y gezeigt.

!G (Go)
Dieser Befehl startet die Vorführung des Films.

!H (Halt)
stoppt die Vorführung des Films.

!C,x[,y] (Color)
Dieser Befehl legt die Bildschirmfarben des C64 fest. »x« und »y« liegen zwischen 0 und 15, wobei x die Rahmenfarbe und y die Hintergrundfarbe angibt. Die Eingabe von »y« ist nicht unbedingt notwendig. Ein Ändern der Rahmenfarbe durch POKE 53280, x ist nicht möglich.

!O (OFF)
Dieser Befehl schaltet die Basic-Erweiterung ab. Mit SYS 49152 kann man sie wieder aktivieren.

Neben den neuen Befehlen wurde auch eine neue Fehlermeldung eingebaut. Sie lautet »?IRQ CONFLICT ERROR« und bedeutet, daß während der Vorführung eines Films versucht wurde, auf Diskette oder Datasette zuzugreifen, was durch die Überschneidung von Interrupts zum Absturz des C64, oder zumindest zu störendem Flimmern führen würde. Dieser Fehler tritt ebenso auf, wenn versucht wird, einen Film mit »!G« zu starten, solange eine Datei geöffnet ist. Dies ist notwendig, da das Schreiben in eine Datei während der Filmvorführung ebenfalls einen Absturz des Computers oder ein Flimmern des Bildes zur Folge hätte.

Soll die MBB-Basic-Erweiterung von eigenen Programmen geladen werden, so muß der Programmkopf, wie folgt lauten:

```
10 A=A + 1:IF A=1 THEN LOAD "MBB-IRQ.CDE",8,1
20 SYS 49152: REM BASIC-ERWEITERUNG INITIALISIEREN
30 SYS 50009: REM COPYRIGHT
```

Magic Border Beams für Assembler-Freaks

Selbstverständlich können die Routinen von MBB auch in Maschinensprache genutzt werden. Für die Assembler-Spezialisten hier noch die Programm-Sequenzen, die nötig sind, um die MBB-Befehle von »MBB-IRQ.CDE« in eigenen Maschinenprogrammen anzusteuern.

Für den »!L«-Befehl gilt folgender Aufruf:

```
LDA #$ x ; x = Länge des Filenamens
LDX #$ y ; y = LSB der Adresse des Filenamens
LDY #$ z ; z = MSB der Adresse des Filenamens
JSR $ffbd ; Fileparameter setzen
JSR $c27e ; Film laden, entpacken und Parameter
            setzen
```

Beim »!S«-Befehl:

```
LDX #$ ?; Filmgeschwindigkeit zwischen $00 und $20
JSR $c1c7; Geschwindigkeit setzen
```

»!M«-Befehl:

```
LDA #$ x ; x = erstes zu zeigendes Bild
STA $fd  ; Zwischenspeichern
LDX #$ y ; y= letztes zu zeigendes Bild
JSR $c1af; Filmparameter setzen
```

»!G«-Befehl:

```
JSR $c003; Film starten
```

»!H«-Befehl:

```
JSR $c1d4; Film stoppen
```

»!C«-Befehl:

```
LDX #$ r; r = Code der Rahmenfarbe zwischen $00
          und $0f
LDY #$ b; b = Code der Bildschirmfarbe zwischen
          $00 und $0f
STX $d020;
STX $c400;
STX $c440;
STY $d021; In Register schreiben
```

Bei der Anwendung von MBB in Maschinensprache ist ebenfalls zu beachten, daß Operationen mit einem Diskettenlaufwerk oder der Datasette während der Filmvorführung zu einem Flimmern des Monitors oder einem Absturz des C64 führen können.

Abschließend die Speicherbereiche, die von »Magic Border Beams« belegt werden:

$00FB-$00FF – MBB-Variablen
$0340-$0344 – MBB-Parameter
$A000-$bFFF – MBB-Film
$C000-$C45F – MBB-Routinen

Nun stehen der Arbeit mit »Magic Border Beams« keine Hindernisse mehr im Weg. Zur Einstimmung können Sie unseren Demo-Film in Listing 5 mit dem MSE eingeben. Er läßt sich mit dem Editor weiterbearbeiten und kann auch mit der Basic-Erweiterung in eigenen Programmen verwendet werden.

Aus Platzgründen kann hier nur ein Demonstrationsfilm abgedruckt werden.

Auf der Programmservice-Diskette, die zu diesem Sonderheft erhältlich ist, finden Sie jedoch fünf weitere Beispiele für eine phantasievolle Rahmen-Gestaltung.

Lassen Sie sich von ihrer Wirkung inspirieren, um bald mit eigenen Kreationen den Rahmen Ihres Bildschirms zu beleben.

(Matthias Fichtner/Michael Thomas)

```
1000 REM ****************************** <036>
1010 REM *                            * <041>
1020 REM *     MAGIC BORDER BEAMS     * <113>
1030 REM *                            * <063>
1040 REM * EDITOR  DE LUXE VERSION #5 * <064>
1050 REM *                            * <083>
1060 REM *    BY MATTHIAS FICHTNER    * <213>
1070 REM *                            * <103>
1080 REM ****************************** <118>
1090 POKE 55,0:POKE 56,120:CLR:POKE 650,25
     5:DIM M(7,24):MM=32769:POKE 788,52   <166>
1100 F(0)=0:F(1)=0:F(2)=128:F(3)=128:SP=0:
     FP=0:LP=0:TP=255:ME=0                <200>
1110 IF PEEK(789)<>234 THEN SYS 49620     <158>
1120 REM ****************************** <158>
1130 REM * INTRO                      * <063>
1140 REM ****************************** <178>
1150 PRINT"{CLR,GRAPHIC,CTRL-H}":POKE 5328
     0,15:POKE 53265,11:POKE 53281,0      <250>
1160 PRINT"{HOME,2DOWN,19SPACE,RED}V"     <198>
1170 PRINT"{18SPACE}VVV"                  <030>
1180 PRINT"{17SPACE}VV{LIG.RED}V{RED}VV"  <167>
1190 PRINT"{16SPACE}VV{LIG.RED}VVV{RED}VV" <214>
1200 PRINT"{15SPACE}VV{LIG.RED}VV{GREY 3}V
     {LIG.RED}VV{RED}VV"                  <117>
1210 PRINT"{14SPACE,BLUE}V{SPACE,RED}VV{LI
     G.RED}VVV{RED}VV{SPACE,GREY 1}V"     <102>
1220 PRINT"{13SPACE,BLUE}VVV{SPACE,RED}VV{
     LIG.RED}V{RED}VV{SPACE,GREY 1}VVV"   <141>
1230 PRINT"{12SPACE,BLUE}VV{LIG.BLUE}V{BLU
     E}VV{SPACE,RED}VVV{SPACE,GREY 1}VV{GR
     EY 2}V{GREY 1}VV"                    <254>
1240 PRINT"{11SPACE,BLUE}VV{LIG.BLUE}VVV{B
     LUE}VV{SPACE,RED}V{SPACE,GREY 1}VV{GR
     EY 2}VVV{GREY 1}VV"                  <232>
1250 PRINT"{10SPACE,BLUE}VV{LIG.BLUE}VV{GR
     EY 3}V{LIG.BLUE}VV{BLUE}VV{SPACE,GREY
      1}VV{GREY 2}VV{GREY 3}V{GREY 2}VV{GR
     EY 1}VV"                             <212>
1260 PRINT"{11SPACE,BLUE}VV{LIG.BLUE}VVV{B
     LUE}VV{3SPACE,GREY 1}VV{GREY 2}VVV{GR
     EY 1}VV"                             <094>
1270 PRINT"{12SPACE,BLUE}VV{LIG.BLUE}V{BLU
     E}VV{5SPACE,GREY 1}VV{GREY 2}V{GREY 1
     }VV"                                 <243>
1280 PRINT"{13SPACE,BLUE}VVV{7SPACE,GREY 1
     }VVV"                                <227>
1290 PRINT"{14SPACE,BLUE}V{SPACE,GREY 3}T
     R A X{SPACE,GREY 1}V{DOWN}"          <110>
1300 PRINT"{12SPACE,GREY 3}S O F T W A R E
     {5DOWN}"                             <068>
1310 POKE 53265,27                        <059>
1320 N$="MBB-EDI.ASS":A=30720:GOSUB 2970  <106>
1330 N$="MBB-IRQ.CDE":A=49152:GOSUB 2970  <198>
1340 C(0)=0:C(1)=11:C(2)=12:C(3)=15:C(4)=1 <111>
1350 READ A$:IF A$="*"THEN RESTORE:GOTO 13
     50                                   <158>
1360 FOR T=0 TO 4                         <118>
1370 POKE 646,C(T):PRINT"{UP}"A$          <072>
1380 FOR I=1 TO 30:NEXT I,T:T=0           <223>
1390 GET B$:IF B$=CHR$(13)THEN 1540       <007>
1400 T=T+1:IF T<600 THEN 1390             <012>
1410 FOR T=4 TO 0 STEP-1                  <234>
1420 POKE 646,C(T):PRINT"{UP}"A$          <122>
1430 FOR I=1 TO 30:NEXT I,T               <071>
1440 GOTO 1350                            <062>
1450 DATA"{7SPACE}MAGIC BORDER BEAMS EDITO
     R"                                   <228>
1460 DATA"{10SPACE}DE LUXE VERSION #05"   <146>
1470 DATA"{3SPACE}WRITTEN 1987 BY MATTHIAS
      FICHTNER"                           <027>
1480 DATA"{7SPACE}(C) 1987 BY TRAX SOFTWAR
     E"                                   <069>
1490 DATA"{10SPACE}HIT RETURN TO ENTER"   <092>
1500 DATA"*"                              <235>
1510 REM ****************************** <038>
1520 REM * INIT SCREEN                * <085>
1530 REM ****************************** <058>
1540 POKE 53265,11:POKE 780,15:SYS 30723  <032>
1550 PRINT"{CLR}":FOR Y=0 TO 23:PRINT"{SPA
     CE,RVSON}@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
     @@@@@@@@@@";                         <196>
1560 POKE 1064+Y*40+39,228:IF Y<23 THEN PR
     INT                                  <196>
1570 NEXT Y                               <002>
1580 P=0:L=0                              <244>
1590 REM ****************************** <120>
1600 REM * MAIN                       * <122>
1610 REM ****************************** <140>
1620 PRINT"{HOME,RVOFF,SPACE,GREY 1}Z{GREY
      2}Z{GREY 3}Z MAGIC BORDER BEAMS Z{GR
     EY 2}Z{GREY 1}Z{4SPACE,GREY 3}PIC: "R
     IGHT$("{2SPACE}"+STR$(P),3)          <183>
```

Listing 1. Der komfortable Editor von MBB: »MBB-EDI.SCR«. Bitte mit dem Checksummer eingeben.

```
1630 N=253:GOSUB 2780:SYS 30720                  <233>
1640 POKE 211,0:POKE 214,24-L:SYS 58732          <248>
1650 PRINT">";:POKE 53265,27                     <201>
1660 GOSUB 2820                                  <036>
1670 N=(P-(A$="+")+(A$="-")-(A$="±")*10+(A
     $="_")*10)AND 255                           <089>
1680 IF N<>P THEN P=N:GOTO 1620                  <231>
1690 IF A$="{DOWN}"AND L>0 THEN PRINT"{LEF
     T,SPACE,DOWN,LEFT}>";:L=L-1:GOTO 1660       <232>
1700 IF A$="{UP}"AND L<23 THEN PRINT"{LEFT
     ,SPACE,UP,LEFT}>";:L=L+1:GOTO 1660          <235>
1710 IF A$="U"THEN N=251:GOSUB 2780:SYS 30
     732:GOTO 1620                               <013>
1720 IF A$="D"THEN N=251:GOSUB 2780:SYS 30
     735:GOTO 1620                               <167>
1730 IF A$="="AND P>0 THEN 3680                  <137>
1740 T=1                                         <209>
1750 IF A$=MID$("{BLACK,WHITE,RED,CYAN,PUR
     PLE,GREEN,BLUE,YELLOW,ORANGE,BROWN,LI
     G.RED,GREY 1,GREY 2,LIG.GREEN,LIG.BLU
     E,GREY 3}",T,1)THEN POKE MM+P*32+L,T-
     1:GOTO 1630                                 <091>
1760 T=T+1:IF T<17 THEN 1750                     <066>
1770 IF A$="B"THEN POKE 53280,(PEEK(53280)
     +1)AND 15:GOTO 1660                         <252>
1780 IF A$="{CLR}"THEN FOR T=0 TO 23:POKE
     MM+P*32+T,PEEK(53280)AND 15:NEXT:GOTO
     1630                                        <229>
1790 IF A$="M"THEN 3240                          <031>
1800 IF A$="R"THEN 3480                          <254>
1810 IF A$="L"THEN 1990                          <053>
1820 IF A$="S"THEN 2070                          <094>
1830 IF A$="Q"THEN 2310                          <031>
1840 IF A$="C"THEN 2370                          <130>
1850 IF A$="T"THEN 2620                          <221>
1860 IF A$="N"THEN 2860                          <052>
1870 IF A$="G"THEN 3130                          <086>
1880 IF A$="I"THEN 3610                          <105>
1890 IF A$="P"THEN 3720                          <015>
1900 IF A$="F"THEN 4040                          <000>
1910 T=1                                         <125>
1920 IF A$=MID$("{F2,F4,F6,F8}",T,1)THEN F
     (T-1)=P                                     <177>
1930 IF A$=MID$("{F1,F3,F5,F7}",T,1)THEN P
     =F(T-1):GOTO 1620                           <166>
1940 T=T+1:IF T<5 THEN 1920                      <108>
1950 GOTO 1660                                   <144>
1960 REM ******************************          <236>
1970 REM * LOAD PIC                    *         <174>
1980 REM ******************************          <000>
1990 GOSUB 2930:PRINT"{HOME,SPACE}LOAD: ";       <042>
2000 GOSUB 2180:IF LEN(N$)=6 THEN 1620           <175>
2010 A=MM-1:GOSUB 2970:GOSUB 3050:IF N<>0
     THEN 1620                                   <137>
2020 TP=PEEK(36864):FP=PEEK(36865):LP=PEEK
     (36866):SP=PEEK(36867):SYS 30747            <090>
2030 GOTO 1620                                   <096>
2040 REM ******************************          <060>
2050 REM * SAVE PIC                    *         <117>
2060 REM ******************************          <082>
2070 GOSUB 2930:PRINT"{HOME,SPACE}SAVE: ";       <182>
2080 GOSUB 2180:IF LEN(N$)=6 THEN 1620           <235>
2090 FOR T=1 TO LEN(N$):POKE 51711+T,ASC(M
     ID$(N$,T,1)):NEXT                           <111>
2100 POKE 781,8:POKE 782,0:SYS 65466             <170>
2110 POKE 780,LEN(N$):POKE 781,0:POKE 782,
     202:SYS 65469                               <151>
2120 SYS 30744:POKE 36864,TP:POKE 36865,FP
     :POKE 36866,LP:POKE 36867,SP                <083>
2130 POKE 251,0:POKE 252,128:POKE 780,251:
     POKE 781,4:POKE 782,144:SYS 65496           <230>
2140 SYS 30747:GOSUB 3050:GOTO 1620              <067>
2150 REM ******************************          <172>
2160 REM * GET PIC-NAME                *         <128>
2170 REM ******************************          <192>
2180 N$="@:MBB.":PRINT CHR$(34)"MBB.";:POK
     E 212,0:PRINT"{RVSON}@{RVOFF,LEFT}";:
     FOR T=1 TO 12                               <101>
2190 GOSUB 2820                                  <058>
2200 IF(A$<" "OR A$>"←"OR A$=CHR$(34))AND
     A$<>CHR$(13)AND A$<>CHR$(20)THEN 2190       <069>
2210 IF A$=CHR$(13)THEN T=13                     <127>
2220 IF A$=CHR$(20)AND T<2 THEN 2190             <087>
2230 IF A$=CHR$(20)THEN PRINT"{SPACE,2LEFT
     ,RVSON}@{RVOFF,LEFT}";:T=T-2:N$=LEFT$
     (N$,LEN(N$)-1):GOTO 2250                    <147>
2240 IF A$<>CHR$(13)THEN PRINT A$"{RVSON}@
     {RVOFF,LEFT}";:N$=N$+A$                     <252>
2250 NEXT                                        <228>
2260 PRINT CHR$(34);:POKE 212,0                  <043>
2270 RETURN                                      <040>
2280 REM ******************************          <046>
2290 REM * QUIT                        *         <246>
2300 REM ******************************          <066>
2310 GOSUB 2930:PRINT"{HOME,SPACE}ARE YOU
     SURE ? ";                                   <064>
2320 N$="N":GOSUB 3990:IF N$="N"THEN 1620        <217>
2330 POKE 24576,0:SYS 64738                      <219>
2340 REM ******************************          <108>
2350 REM * CHANGE COLOR                *         <049>
2360 REM ******************************          <128>
2370 GOSUB 2930:PRINT"{HOME,SPACE}CHANGE <
     >{2LEFT}";:N=PEEK(MM+P*32+L):GOSUB 3
     910:C1=N                                    <173>
2380 PRINT"{2RIGHT,SPACE}INTO < >{2LEFT}";
     :GOSUB 3910:C2=N                            <072>
2390 PRINT"{2RIGHT,5SPACE}IN PIC ";:N=P:GO
     SUB 2490:S1=N                               <193>
2400 PRINT"{3RIGHT}-";                           <210>
2410 N=S1:GOSUB 2490:IF N<S1 THEN 2410           <209>
2420 S2=N:POKE 780,C1:POKE 2,C2:I=P              <244>
2430 N=251:P=S1:GOSUB 2780                       <046>
2440 N=253:P=S2+1:GOSUB 2780                     <210>
2450 SYS 30726:P=I:GOTO 1620                     <002>
2460 REM ******************************          <228>
2470 REM * GET NUMBER OF PIC           *         <255>
2480 REM ******************************          <248>
2490 PX=PEEK(211):PY=PEEK(214)                   <061>
2500 POKE 211,PX:POKE 214,PY:SYS 58732:PRI
     NT RIGHT$("{2SPACE}"+STR$(N),3);            <044>
2510 POKE 254,(MM+N*32)/256:POKE 253,(MM+N
     *32)-PEEK(254)*256:SYS 30720                <195>
2520 GOSUB 2820                                  <134>
2530 X=(N-(A$="+")+(A$="-")-(A$="±")*10+(A
     $="_")*10)AND 255                           <162>
2540 FOR T=1 TO 4:IF A$=MID$("{F1,F3,F5,F7
     }",T,1)THEN X=F(T-1)                        <209>
2550 NEXT                                        <018>
2560 IF X<>N THEN N=X:GOTO 2500                  <051>
2570 IF A$<>CHR$(13)THEN 2520                    <082>
2580 POKE 211,PX:POKE 214,PY:SYS 58732:RET
     URN                                         <159>
2590 REM ******************************          <104>
2600 REM * TRANSFER PICS               *         <040>
2610 REM ******************************          <124>
2620 GOSUB 2930:PRINT"{HOME,SPACE}SOURCE:
     ";                                          <233>
2630 N=P:GOSUB 2490:S1=N                         <085>
2640 PRINT"{3RIGHT}-";                           <196>
2650 N=S1:GOSUB 2490:S2=N:IF S2<S1 THEN 26
     50                                          <250>
2660 PRINT"{3RIGHT,7SPACE}DESTINATION: ";        <174>
2670 N=S1:GOSUB 2490:D=N:I=P                     <020>
2680 AW=0:EW=S2-S1:SW=1:IF D>S1 AND D<=S2
     THEN AW=EW:EW=0:SW=-1                       <222>
2690 FOR T=AW TO EW STEP SW                      <096>
2700 IF(T+D)>255 THEN 2740                       <183>
2710 N=251:P=T+S1:GOSUB 2780                     <181>
2720 N=253:P=T+D:GOSUB 2780                      <253>
2730 SYS 30729                                   <226>
2740 NEXT:P=I:GOTO 1620                          <196>
2750 REM ******************************          <008>
2760 REM * POKE PIC-ADR TO N/N+1       *         <250>
2770 REM ******************************          <028>
2780 POKE N+1,(MM+P*32)/256:POKE N,(MM+P*3
     2)-PEEK(N+1)*256:RETURN                     <159>
2790 REM ******************************          <048>
2800 REM * GETKEY A$                   *         <040>
2810 REM ******************************          <068>
2820 GET A$:ON-(A$="")GOTO 2820:RETURN           <014>
2830 REM ******************************          <090>
2840 REM * INIT FOR NEW MOVIE          *         <187>
2850 REM ******************************          <110>
2860 GOSUB 2930:PRINT"{HOME,SPACE}ARE YOU
     SURE ? ";                                   <106>
2870 N$="N":GOSUB 3990:IF N$="N"THEN 1620        <003>
2880 POKE 780,PEEK(53280)AND 15:SYS 30723:
     FOR T=0 TO 7:M(T,24)=0:NEXT                 <162>
2890 TP=255:FP=0:LP=0:SP=0:GOTO 1580             <071>
2900 REM ******************************          <160>
```

```
2910 REM * CLEAR CURSOR/HEADLINE       *    <254>
2920 REM *******************************    <180>
2930 PRINT"{LEFT,SPACE,HOME,39SPACE}";:RET
     URN                                    <182>
2940 REM *******************************    <200>
2950 REM * LOAD N$ TO A                *    <004>
2960 REM *******************************    <220>
2970 FOR T=1 TO LEN(N$):POKE 51711+T,ASC(M
     ID$(N$,T,1)):NEXT                      <229>
2980 POKE 781,8:POKE 782,0:SYS 65466        <032>
2990 POKE 780,LEN(N$):POKE 781,0:POKE 782,
     202:SYS 65469                          <013>
3000 POKE 780,0:POKE 782,A/256:POKE 781,A-
     PEEK(782)*256:SYS 65493                <041>
3010 RETURN                                 <018>
3020 REM *******************************    <024>
3030 REM * CHECK DISK STATUS           *    <171>
3040 REM *******************************    <044>
3050 OPEN 1,8,15:INPUT#1,N:CLOSE 1          <081>
3060 IF N=0 THEN RETURN                     <011>
3070 PRINT"{HOME,39SPACE}";                 <206>
3080 PRINT"{HOME,SPACE}I/O-ERROR...";       <051>
3090 GOSUB 2820:RETURN                      <113>
3100 REM *******************************    <106>
3110 REM * SHOW THE MOVIE              *    <200>
3120 REM *******************************    <126>
3130 PRINT"{LEFT,SPACE}";:POKE 832,SP:POKE
      833,1                                 <230>
3140 POKE 834,FP:POKE 252,(FP-1)AND 255     <204>
3150 POKE 835,LP:CC=PEEK(53280)AND 15:POKE
      50176,CC:POKE 50240,CC                <123>
3160 POKE 836,TP                            <013>
3170 SYS 30738:SYS 49155                    <116>
3180 GOSUB 2820:IF A$="B"THEN CC=(CC+1)AND
      15:POKE 50176,CC:POKE 50240,CC        <153>
3190 IF A$<>"H"THEN 3180                    <228>
3200 SYS 49620:GOTO 1620                    <155>
3210 REM *******************************    <216>
3220 REM * MEMORIZE                    *    <202>
3230 REM *******************************    <236>
3240 PX=PEEK(211):PY=PEEK(214)              <049>
3250 GOSUB 2930:PRINT"{HOME,SPACE}MEMORY:"
     ;                                      <109>
3260 PRINT ME"{3LEFT}";                     <062>
3270 GOSUB 2820                             <122>
3280 ME=(ME-(A$="+")+(A$="-"))AND 7         <189>
3290 IF A$="←"THEN 1620                     <008>
3300 IF A$<>CHR$(13)THEN 3260               <145>
3310 POKE 211,PX:POKE 214,PY:SYS 58732:PRI
     NT"{LEFT}>";:B=L                       <241>
3320 GOSUB 2820                             <172>
3330 IF A$="{DOWN}"AND B>0 THEN PRINT"{LEF
     T,SPACE,DOWN,LEFT}>";:B=B-1            <076>
3340 IF A$="{UP}"AND B<23 THEN PRINT"{LEFT
     ,SPACE,UP,LEFT}>";:B=B+1               <100>
3350 IF A$<>CHR$(13)THEN 3320               <003>
3360 T=B                                    <145>
3370 GOSUB 2820                             <224>
3380 IF A$="{DOWN}"AND T>B THEN PRINT"{LEF
     T,SPACE,DOWN}";:T=T-1                  <076>
3390 IF A$="{UP}"AND T<23 THEN PRINT"{UP,L
     EFT}>";:T=T+1                          <159>
3400 IF A$<>CHR$(13)THEN 3370               <183>
3410 FOR I=0 TO T-B:M(ME,I)=PEEK(MM+32*P+I
     +B):NEXT                               <177>
3420 M(ME,24)=T-B+1:PRINT"{LEFT,SPACE}";    <044>
3430 IF T>B THEN PRINT"{DOWN,LEFT,SPACE}";
     :T=T-1:GOTO 3430                       <175>
3440 GOTO 1620                              <238>
3450 REM *******************************    <202>
3460 REM * RECALL                      *    <246>
3470 REM *******************************    <222>
3480 GOSUB 2930:PRINT"{HOME,SPACE}MEMORY:"
     ;                                      <085>
3490 PRINT ME"{3LEFT}";                     <038>
3500 GOSUB 2820                             <098>
3510 ME=(ME-(A$="+")+(A$="-"))AND 7         <165>
3520 IF A$="←"THEN 1620                     <240>
3530 IF A$<>CHR$(13)THEN 3490               <123>
3540 IF M(ME,24)=0 THEN 3570                <223>
3550 FOR T=0 TO M(ME,24)-1:IF L+T<24 THEN
     POKE MM+P*32+L+T,M(ME,T)               <022>
3560 NEXT                                   <012>
3570 GOTO 1620                              <112>
3580 REM *******************************    <076>
3590 REM * INVERT ORDER                *    <163>
3600 REM *******************************    <098>
3610 GOSUB 2930:PRINT"{HOME,SPACE}INVERT O
     RDER OF PIC ";                         <094>
3620 N=P:GOSUB 2490:P1=N:PRINT"{3RIGHT}-"; <243>
3630 N=P1:GOSUB 2490:IF N<P1 THEN 3630      <212>
3640 P2=N:I=P:FOR T=0 TO INT((P2-P1)/2)     <139>
3650 N=251:P=P1+T:GOSUB 2780                <102>
3660 N=253:P=P2-T:GOSUB 2780                <008>
3670 SYS 30741:NEXT:P=I:GOTO 1620           <072>
3680 REM *******************************    <178>
3690 REM * COPY PREVIOUS PIC           *    <104>
3700 REM *******************************    <198>
3710 N=251:P=P-1:GOSUB 2780:N=253:P=P+1:GO
     SUB 2780:SYS 30729:GOTO 1620           <175>
3720 REM *******************************    <218>
3730 REM * GET PARAMETERS              *    <233>
3740 REM *******************************    <238>
3750 GOSUB 2930:PRINT"{HOME,SPACE}SPEED: "
     ;                                      <248>
3760 PRINT RIGHT$(STR$(SP),2)"{2LEFT}";     <211>
3770 GOSUB 2820                             <114>
3780 SP=(SP-(A$="+")+(A$="-"))AND 31        <018>
3790 IF A$<>CHR$(13)THEN 3760               <192>
3800 PRINT"{HOME,SPACE}MOVIE: ";            <194>
3810 N=FP:GOSUB 2490:FP=N                   <179>
3820 PRINT"{3RIGHT}-";:N=(LP-1)AND 255:GOS
     UB 2490:LP=(N+1)AND 255                <008>
3830 PRINT"{HOME,SPACE}TRANSPARENT COLOR ?
      ";                                    <254>
3840 N$="Y":IF TP=255 THEN N$="N":TP=PEEK(
     MM+P*32+L)                             <236>
3850 GOSUB 3990                             <189>
3860 IF N$="N"THEN TP=255:GOTO 1620         <160>
3870 PRINT"{3LEFT}: < >{2LEFT}";:N=TP:GOSU
     B 3910:TP=N:GOTO 1620                  <078>
3880 REM *******************************    <124>
3890 REM * GET COLOR                   *    <009>
3900 REM *******************************    <144>
3910 POKE 646,N:PRINT"{RVSON}@{RVOFF,LEFT,
     GREY 3}";                              <155>
3920 GOSUB 2820                             <010>
3930 N=(N-(A$="+")+(A$="-"))AND 15          <047>
3940 IF A$<>CHR$(13)THEN 3910               <086>
3950 RETURN                                 <198>
3960 REM *******************************    <204>
3970 REM * GET Y OR N                  *    <137>
3980 REM *******************************    <224>
3990 PRINT N$"{LEFT}";                      <197>
4000 GOSUB 2820                             <090>
4010 IF A$="Y"OR A$="N"THEN N$=A$           <136>
4020 IF A$<>CHR$(13)THEN 3990               <170>
4030 RETURN                                 <022>
4040 REM *******************************    <028>
4050 REM * TRANSFER PICS FROM FILE     *    <175>
4060 REM *******************************    <048>
4070 GOSUB 2930:PRINT"{HOME,SPACE}TRANSFER
      FROM FILE: ";                         <096>
4080 GOSUB 2180:IF LEN(N$)=6 THEN 1620      <223>
4090 A=40960:GOSUB 2970:GOSUB 3050:IF N<>0
      THEN 1620                             <176>
4100 SYS 49893:MM=40961                     <065>
4110 GOSUB 2930:PRINT"{HOME,SPACE}SOURCE:
     ";                                     <199>
4120 N=0:GOSUB 2490:S1=N                    <049>
4130 PRINT"{3RIGHT}-";                      <162>
4140 N=S1:GOSUB 2490:S2=N:IF S2<S1 THEN 41
     40                                     <192>
4150 MM=32769:PRINT"{3RIGHT,7SPACE}DESTINA
     TION: ";                               <177>
4160 N=P:GOSUB 2490:D=N:I=P                 <102>
4170 AW=0:EW=S2-S1:SW=1:IF D>S1 AND D<=S2
     THEN AW=EW:EW=0:SW=-1                  <188>
4180 FOR T=AW TO EW STEP SW                 <062>
4190 IF(T+D)>255 THEN 4230                  <019>
4200 MM=40961:N=251:P=T+S1:GOSUB 2780       <028>
4210 MM=32769:N=253:P=T+D:GOSUB 2780        <015>
4220 SYS 30729                              <192>
4230 NEXT:P=I:GOTO 1620                     <162>
```

Listing 1. (Schluß)

```
Name : mbb-edi.ass           7800 794d
--------------------------------------
7800 : 4c 1e 78 4c 4e 78 4c 66   aa
7808 : 78 4c 85 78 4c 97 78 4c   13
7810 : a9 78 4c bd 78 4c df 78   1a
7818 : 4c ef 78 4c 1f 79 a9 36   d4
7820 : 85 01 a9 13 20 d2 ff a0   cc
7828 : 17 a9 11 20 d2 ff a5 d1   c4
7830 : 85 fb a5 d2 18 69 d4 85   a2
7838 : fc b1 fd 84 02 a0 27 91   02
7840 : fb 88 d0 fb a4 02 88 10   cf
7848 : e0 a9 37 85 01 60 a2 00   19
7850 : a0 80 86 fb 84 fc a0 00   04
7858 : 91 fb c8 d0 fb e6 fc a6   6b
7860 : fc e0 a0 90 f3 60 a0 00   cb
7868 : d1 fb d0 06 48 a5 02 91   09
7870 : fb 68 e6 fb d0 02 e6 fc   8b
7878 : a6 fc e4 fe 90 ea a6 fb   a8
7880 : e4 fd 90 e4 60 a2 36 86   24
7888 : 01 a0 17 b1 fb 91 fd 88   2b
7890 : 10 f9 a9 37 85 01 60 a0   11
7898 : 17 b1 fb aa 88 b1 fb c8   74
78a0 : 91 fb 88 d0 f7 8a 91 fb   7d
78a8 : 60 a0 00 b1 fb aa c8 b1   2a
78b0 : fb 88 91 fb c8 c0 17 d0   64
78b8 : f5 8a 91 fb 60 a2 00 a0   32
78c0 : 80 86 fb 84 fc a0 a0 86   77
78c8 : fd 84 fe a0 00 b1 fb 91   7c
78d0 : fd c8 d0 f9 e6 fc e6 fe   94
78d8 : a5 fc c9 a0 90 ef 60 a0   cd
78e0 : 17 b1 fb 48 b1 fd 91 fb   21
78e8 : 68 91 fd 88 10 f3 60 a2   11
78f0 : 00 a0 80 86 fb 84 fc a0   4a
78f8 : 90 86 fd 84 fe a0 00 b1   34
7900 : fb 29 0f 0a 0a 0a 0a 91   d1
7908 : fb b1 fd 29 0f 11 fb 91   0d
7910 : fb c8 d0 eb e6 fc e6 fe   11
7918 : a5 fe c9 a0 d0 e1 60 a2   a6
7920 : 00 a0 80 86 fb 84 fc a0   7a
7928 : 90 86 fd 84 fe a0 00 b1   64
7930 : fb 29 0f 91 fd b1 fb 29   65
7938 : f0 4a 4a 4a 4a 91 fb c8   dc
7940 : d0 ed e6 fc e6 fe a5 fe   5b
7948 : c9 a0 d0 e3 60 ff 00 01   1a
```

Listing 2. »MBB-EDI.ASS« enthält Hilfsroutinen für den Editor. Bitte mit dem MSE eingeben.

```
Name : mbb-irq.cde           c000 c3a8
--------------------------------------
c000 : 4c f4 c1 a5 98 f0 05 a2   56
c008 : 00 4c be c2 78 a2 4e a0   4d
c010 : c0 8e 14 03 8c 15 03 a9   4e
c018 : fc 8d 12 d0 ad 11 d0 29   72
c020 : 7f 8d 11 d0 a9 01 8d 1a   d1
c028 : d0 a9 7f 8d 0d dc ad 0d   e7
c030 : dc a9 00 85 fb 8d 41 03   c9
c038 : ae 42 03 ca 86 fc ad 20   69
c040 : d0 a2 18 9d 00 c4 9d 40   38
c048 : c4 ca 10 f7 58 60 ad 19   e6
c050 : d0 8d 19 d0 29 01 d0 03   2b
c058 : 4c f1 c0 a5 fb 49 01 85   9b
c060 : fb f0 47 ea ea ea ea ea   8a
c068 : ea ea ea ea ea ea ea ea   67
c070 : ea ea ea ea ea ea ea ea   6f
c078 : ea ea ea ea ea ea a2 18   b0
c080 : bd 00 c4 8d 20 d0 ea ea   2a
c088 : ea ea ea ea ea ea ea ea   87
c090 : ea ea ea ea ea ea ea ea   8f
c098 : ea ea ea ea ea ea ea ca   56
c0a0 : 10 de a9 fa 8d 12 d0 4c   2e
c0a8 : f1 c0 ea ea ea ea ea ea   99
c0b0 : ea ea ea ea ea ea ea ea   af
c0b8 : ea ea ea ea ea ea ea ea   b7
c0c0 : ea ea ea ea ea a2 18 bd   d7
c0c8 : 40 c4 8d 20 d0 ea ea ea   b8
c0d0 : ea ea ea ea ea ea ea ea   cf
c0d8 : ea ea ea ea ea ea ea ea   d7
c0e0 : ea ea ea ea ea ea ca 10   a8
c0e8 : de a9 18 8d 12 d0 4c f8   1d
c0f0 : c0 68 a8 68 aa 68 58 40   eb
c0f8 : ad 41 03 c9 20 f0 0a ad   4d
c100 : 41 03 f0 0a aa ca 8e 41   fe
c108 : 03 a5 fc 4c 22 c1 ad 40   0e
c110 : 03 8d 41 03 e6 fc a5 fc   71
c118 : cd 43 03 d0 05 ad 42 03   2f
c120 : 85 fc 29 07 aa bd 57 c1   c8
c128 : 85 fd a5 fc 4a 4a 4a 18   05
c130 : 69 a0 85 fe a9 36 85 01   8f
c138 : a0 18 a2 01 b1 fd cd 44   78
c140 : 03 d0 03 ad 20 d0 99 00   11
c148 : c4 9d 40 c4 e8 88 d0 ec   73
c150 : a9 37 85 01 4c 31 ea 00   10
c158 : 20 40 60 80 a0 c0 e0 a9   a7
c160 : 2c a0 00 d1 7a d0 2a 20   2e
c168 : 9b b7 e0 10 b0 26 8e 00   8f
c170 : c4 8e 40 c4 ad 15 03 c9   47
c178 : ea d0 03 8e 20 d0 a9 2c   e4
c180 : a0 00 d1 7a d0 0a 20 9b   f9
c188 : b7 e0 10 b0 07 8e 21 d0   d5
c190 : 60 4c 08 af 4c 48 b2 a9   33
c198 : 2c a0 00 d1 7a d0 f2 20   89
c1a0 : 9b b7 86 fd a9 2c a0 00   f7
c1a8 : d1 7a d0 e5 20 9b b7 e8   37
c1b0 : 8e 43 03 a6 fd 8e 42 03   d9
c1b8 : ca 86 fc 60 a9 2c a0 00   8f
c1c0 : d1 7a d0 cd 20 9b b7 e0   3c
c1c8 : 21 b0 c9 8e 40 03 a9 01   4a
c1d0 : 8d 41 03 60 78 a2 31 a0   6d
c1d8 : ea 8e 14 03 8c 15 03 a9   40
c1e0 : ff 8d 0d dc ad 0d dc ad   97
c1e8 : 00 c4 8d 20 d0 a9 00 8d   27
c1f0 : 1a d0 58 60 a2 27 a0 c2   00
c1f8 : 8e 08 03 8c 09 03 a2 47   9e
c200 : a0 c3 8e 30 03 8c 31 03   8b
c208 : a2 4d a0 c3 8e 32 03 8c   91
c210 : 33 03 a2 53 a0 c3 8e 1a   6e
c218 : 03 8c 1b 03 a9 00 8d 42   de
c220 : 03 8d 43 03 85 fc 60 20   1d
c228 : 73 00 c9 21 f0 06 20 79   e5
c230 : 00 4c e7 a7 20 73 00 48   73
c238 : 20 73 00 68 a2 06 dd 5b   a7
c240 : c2 f0 06 ca 10 f8 4c 08   5f
c248 : af a9 a7 48 a9 ad 48 8a   fd
c250 : 0a aa bd 63 c2 48 bd 62   b5
c258 : c2 48 60 4d 53 47 48 43   17
c260 : 4c 4f 96 c1 bb c1 02 c0   85
c268 : d3 c1 5e c1 6f c2 1a c3   e9
c270 : a9 2c a0 00 d1 7a d0 ce   29
c278 : 20 73 00 20 57 e2 a2 08   7d
c280 : a0 00 20 ba ff a9 00 aa   22
c288 : a0 a0 20 d5 ff 90 05 a2   19
c290 : 1d 4c 37 a4 a9 36 85 01   9a
c298 : ae 00 b0 8e 44 03 ae 01   5d
c2a0 : b0 8e 42 03 ca 86 fc ae   ba
c2a8 : 02 b0 8e 43 03 ad 03 b0   19
c2b0 : 48 20 e5 c2 68 aa 4c cb   7f
c2b8 : c1 68 68 68 a2 00 bd cc   8f
c2c0 : c2 f0 06 20 d2 ff e8 d0   f2
c2c8 : f5 4c 62 a4 0d 3f 49 52   a5
c2d0 : 51 20 43 4f 4e 46 4c 49   c7
c2d8 : 43 54 00 48 ad 15 03 c9   72
c2e0 : ea d0 d6 68 60 a9 36 85   2c
c2e8 : 01 a2 00 a0 a0 86 fb 84   86
c2f0 : fc a0 b0 86 fd 84 fe a0   7a
c2f8 : 00 b1 fb 29 0f 91 fd b1   ce
c300 : fb 29 f0 4a 4a 4a 4a 91   58
c308 : fb c8 d0 ed e6 fc e6 fe   49
c310 : a5 fe c9 c0 d0 e3 a9 37   00
c318 : 85 01 60 a2 e4 a0 a7 8e   99
c320 : 08 03 8c 09 03 a2 a5 a0   0b
c328 : f4 8e 30 03 8c 31 03 a2   73
c330 : ed a0 f5 8e 32 03 8c 33   90
c338 : 03 a2 4a a0 f3 8e 1a 03   55
c340 : 8c 1b 03 20 d4 c1 60 20   3c
c348 : db c2 4c a5 f4 20 db c2   91
c350 : 4c ed f5 20 db c2 4c 4a   ae
c358 : f3 a2 00 bd 68 c3 f0 07   cb
c360 : 20 d2 ff e8 4c 5b c3 60   76
c368 : 0d 4d 41 47 49 43 20 42   09
c370 : 4f 52 44 45 52 20 42 45   5c
c378 : 41 4d 53 20 28 43 29 20   ba
c380 : 31 39 38 37 20 4d 26 54   f1
c388 : 00 67 6b 60 6c 71 68 62   db
c390 : 7a 24 0a 0b 04 6e 07 0f   ee
c398 : 01 08 0a 04 14 77 6f 17   89
c3a0 : 6c 6b 7b 63 31 3d 01 20   4e
```

Listing 3. Das Maschinenprogramm »MBB-IRQ.CDE« mit den nötigen Interrupt-Routinen und der Basic-Erweiterung. Bitte mit dem MSE eingeben.

```
1000 REM ******************************   <036>
1010 REM *                            *   <041>
1020 REM *     MAGIC BORDER BEAMS     *   <113>
1030 REM *                            *   <063>
1040 REM *   INTERRUPT  VERSION #02   *   <126>
1050 REM *                            *   <083>
1060 REM *    BY MATTHIAS FICHTNER    *   <213>
1070 REM *                            *   <103>
1080 REM ******************************   <118>
1090 A=A+1:IF A=2 THEN 1280               <150>
1100 PRINT"{CLR,GRAPHIC,CTRL-H}":POKE 5328
     0,15:POKE 53265,11:POKE 53281,0      <200>
1110 PRINT"{HOME,2DOWN,19SPACE,RED}V"     <148>
1120 PRINT"{18SPACE}VVV"                  <236>
1130 PRINT"{17SPACE}VV{LIG.RED}V{RED}VV"  <117>
1140 PRINT"{16SPACE}VV{LIG.RED}VVV{RED}VV" <164>
1150 PRINT"{15SPACE}VV{LIG.RED}VV{GREY 3}V
     {LIG.RED}VV{RED}VV"                  <067>
1160 PRINT"{14SPACE,BLUE}V{SPACE,RED}VV{LI
     G.RED}VVV{RED}VV{SPACE,GREY 1}V"     <052>
1170 PRINT"{13SPACE,BLUE}VVV{SPACE,RED}VV{
     LIG.RED}V{RED}VV{SPACE,GREY 1}VVV"   <091>
1180 PRINT"{12SPACE,BLUE}VV{LIG.BLUE}V{BLU
     E}VV{SPACE,RED}VVV{SPACE,GREY 1}VV{GR
     EY 2}V{GREY 1}VV"                    <204>
1190 PRINT"{11SPACE,BLUE}VV{LIG.BLUE}VVV{B
     LUE}VV{SPACE,RED}V{SPACE,GREY 1}VV{GR
     EY 2}VVV{GREY 1}VV"                  <182>
1200 PRINT"{10SPACE,BLUE}VV{LIG.BLUE}VV{GR
     EY 3}V{LIG.BLUE}VV{BLUE}VV{SPACE,GREY
      1}VV{GREY 2}VV{GREY 3}V{GREY 2}VV{GR
     EY 1}VV"                             <162>
1210 PRINT"{11SPACE,BLUE}VV{LIG.BLUE}VVV{B
     LUE}VV{3SPACE,GREY 1}VV{GREY 2}VVV{GR
     EY 1}VV"                             <044>
1220 PRINT"{12SPACE,BLUE}VV{LIG.BLUE}V{BLU
     E}VV{5SPACE,GREY 1}VV{GREY 2}V{GREY 1
     }VV"                                 <193>
1230 PRINT"{13SPACE,BLUE}VVV{7SPACE,GREY 1
     }VVV"                                <175>
1240 PRINT"{14SPACE,BLUE}V{SPACE,GREY 3}T
     R A X{SPACE,GREY 1}V{DOWN}"          <058>
1250 PRINT"{12SPACE,GREY 3}S O F T W A R E
     {5DOWN}"                             <016>
1260 POKE 53265,27                        <007>
1270 LOAD"MBB-IRQ.CDE",8,1                <053>
1280 C(0)=0:C(1)=11:C(2)=12:C(3)=15:C(4)=1 <051>
1290 READ A$:IF A$="*"THEN RESTORE:GOTO 12
     90                                   <126>
1300 FOR T=0 TO 4                         <058>
```

```
1310 POKE 646,C(T):PRINT"{UP}"A$                    <012>
1320 FOR I=1 TO 20:NEXT I,T:T=0                     <035>
1330 GET B$:IF B$=CHR$(13)THEN 1450                 <205>
1340 T=T+1:IF T<90 THEN 1330                        <104>
1350 FOR T=4 TO 0 STEP-1                            <174>
1360 POKE 646,C(T):PRINT"{UP}"A$                    <062>
1370 FOR I=1 TO 20:NEXT I,T                         <139>
1380 GOTO 1290                                      <115>
1390 DATA"{5SPACE}MAGIC BORDER BEAMS{2SPAC
     E}INTERRUPT"                                   <063>
1400 DATA"{14SPACE}VERSION #02"                     <245>
1410 DATA"{3SPACE}WRITTEN 1987 BY MATTHIAS
     FICHTNER"                                      <223>
1420 DATA"{7SPACE}(C) 1987 BY TRAX SOFTWAR
     E"                                             <009>
1430 DATA"{3SPACE}HIT RETURN TO USE BASIC
     EXTENSION"                                     <042>
1440 DATA"*"                                        <175>
1450 SYS 49152:PRINT"{CLR}";:FOR T=0 TO 3:
     POKE 2048+T,0:NEXT:NEW                         <172>
```

Listing 4. »MBB-IRQ.BOT«: Das kleine Basic-Programm zum Laden und Starten der Basic-Erweiterung. Bitte mit dem Checksummer eingeben.

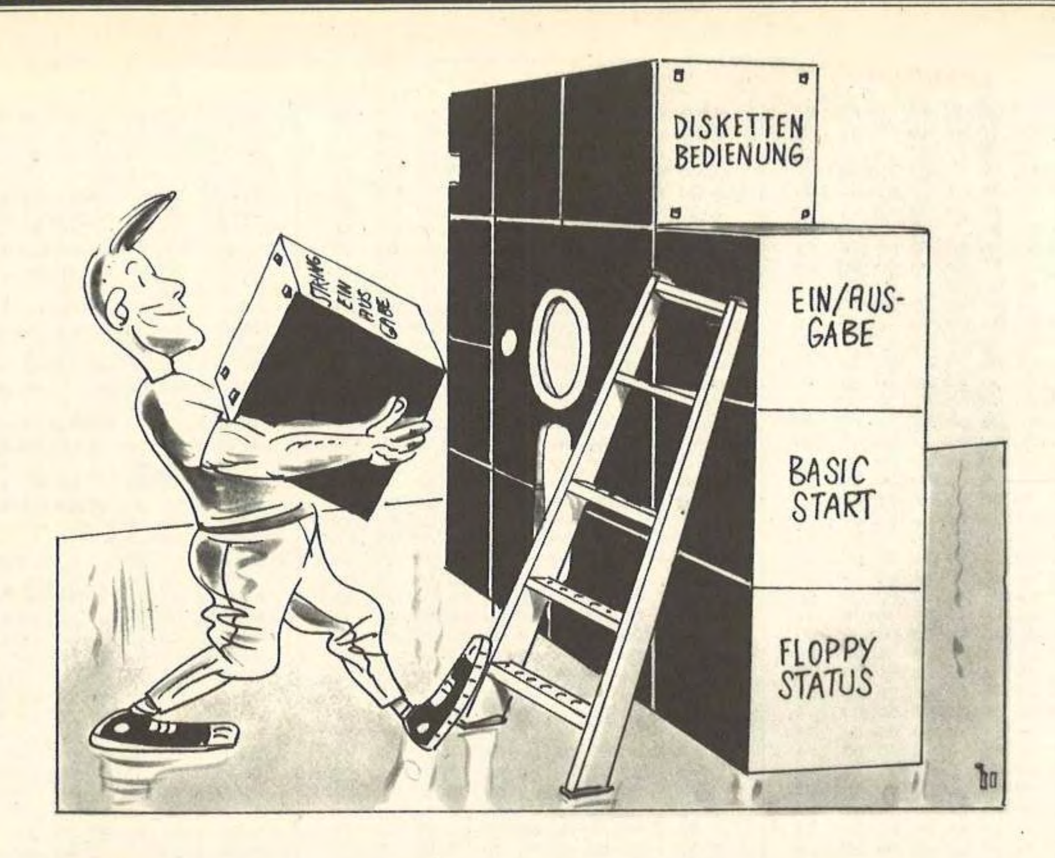
```
Name : mbb.demofilm          8000 9004
--------------------------------------
8000 : 00 60 e0 f0 e0 60 00 00   98
8008 : 00 00 00 06 0e 0f 0e 06   67
8010 : 00 00 00 00 00 00 00 00   11
8018 : 00 00 00 00 00 00 00 00   19
8020 : 00 60 e0 f0 e0 60 b0 00   7a
8028 : 00 00 00 00 06 0e 0f 0e   52
8030 : 06 00 00 00 00 00 00 00   37
8038 : 00 00 00 00 00 00 00 00   39
8040 : 00 60 e0 f0 e0 60 c0 b0   3c
8048 : 00 00 00 00 00 06 0e 0f   cf
8050 : 0e 06 00 00 00 00 00 00   62
8058 : 00 00 00 00 00 00 00 00   59
8060 : 00 60 e0 f0 e0 60 f0 c0   3d
8068 : b0 00 00 00 00 00 06 0e   4d
8070 : 0f 0e 06 00 00 00 00 00   08
8078 : 00 00 00 00 00 00 00 00   79
8080 : 00 60 e0 f0 e0 60 c0 f0   fc
8088 : c0 b0 00 00 00 00 00 06   ad
8090 : 0e 0f 0e 06 00 00 00 00   6a
8098 : 00 00 00 00 00 00 00 00   99
80a0 : 00 60 e0 f0 e0 60 b0 c0   7c
80a8 : f0 c0 b0 00 00 00 00 00   25
80b0 : 06 0e 0f 0e 06 00 00 00   a3
80b8 : 00 00 00 00 00 00 00 00   b9
80c0 : 00 60 e0 f0 e0 60 00 b0   b9
80c8 : c0 f0 c0 b0 00 00 00 00   47
80d0 : 00 06 0e 0f 0e 06 00 00   4a
80d8 : 00 00 00 00 00 00 00 00   d9
80e0 : 00 60 e0 f0 e0 60 00 b0   d9
80e8 : c0 f0 c0 b0 20 00 00 00   69
80f0 : 00 00 06 0e 0f 0e 06 00   ad
80f8 : 00 00 00 00 00 00 00 00   f9
8100 : 00 60 e0 f0 e0 60 00 b0   f9
8108 : c0 f0 c0 b0 a0 20 00 00   92
8110 : 00 00 00 06 0e 0f 0e 06   6f
8118 : 00 00 00 00 00 00 00 00   19
8120 : 00 60 e0 f0 e0 60 00 b0   19
8128 : c0 f0 c0 b0 f0 a0 20 00   3b
8130 : 00 00 00 00 06 0e 0f 0e   5a
8138 : 06 00 00 00 00 00 00 00   3f
8140 : 00 60 e0 f0 e0 60 00 b0   39
8148 : c0 f0 c0 b0 a0 f0 a0 20   1b
8150 : 00 00 00 02 06 0e 0f 0e   ba
8158 : 06 00 00 00 00 00 00 00   5f
8160 : 00 60 e0 f0 e0 60 00 b0   59
8168 : c0 f0 c0 b0 20 a0 f0 a0   f3
8170 : 20 00 02 0a 06 0e 0f 0e   7b
8178 : 06 00 00 00 00 00 00 00   7f
8180 : 00 60 e0 f0 e0 60 00 b0   79
8188 : c0 f0 c0 b0 00 20 a0 f0   6c
8190 : a0 22 0a 0f 06 0e 0f 0e   cf
8198 : 06 00 00 00 00 00 00 00   9f
81a0 : 00 00 60 e0 f0 e0 60 00   6c
81a8 : b0 c0 f0 c0 b0 00 20 a0   d9
81b0 : f2 aa 2f 0a 06 0e 0f 0e   2e
81b8 : 06 00 00 00 00 00 00 00   bf
81c0 : 00 00 00 60 e0 f0 e0 60   a6
81c8 : 00 b0 c0 f0 c0 b0 00 22   44
81d0 : aa ff aa 22 06 0e 0f 0e   92
81d8 : 06 00 00 00 00 00 00 00   df
81e0 : 00 00 00 00 60 e0 f0 e0   73
81e8 : 60 00 b0 c0 f0 c0 b2 0a   80
81f0 : 2f aa f2 a0 26 0e 0f 0e   70
81f8 : 06 00 00 00 00 00 00 00   ff
8200 : 00 00 00 00 00 60 e0 f0   69
8208 : e0 60 00 b0 c0 f0 c2 ba   42
8210 : 0f 2a a2 f6 ae 2f 0e 06   64
8218 : 0b 00 00 00 00 00 00 00   24
8220 : 00 00 00 00 00 00 60 e0   64
8228 : f0 e0 60 00 b0 c0 f2 ca   13
8230 : bf 0a 26 ae ff ae 26 0c   7a
8238 : 0b 00 00 00 00 00 00 00   44
8240 : 00 00 00 00 00 00 00 60   01
8248 : e0 f0 e0 60 00 b0 c2 fa   6b
8250 : cf b6 0e 2f ae f6 af 2c   9e
8258 : 0b 00 00 00 00 00 00 00   64
8260 : 00 00 00 00 00 00 00 00   61
8268 : 60 e0 f0 e0 60 00 b2 ca   f7
8270 : f6 ce bf 0e 26 ac ff ac   a0
8278 : 2b 00 00 00 00 00 00 00   a4
8280 : 00 00 00 00 00 00 00 60   41
8288 : e0 f0 e0 60 00 b0 c2 f6   a3
8290 : ce bf 0e 26 ab fc af 2c   40
8298 : 0b 00 00 00 00 00 00 00   a4
82a0 : 00 00 00 00 00 00 60 e0   e4
82a8 : f0 e0 60 00 b0 c0 f6 ce   ab
82b0 : bf 0e 26 a0 fb ac 2f 0c   0e
82b8 : 0b 00 00 00 00 00 00 00   c4
82c0 : 00 00 00 00 00 60 e0 f0   29
82c8 : e0 60 00 b0 c0 f6 ce bf   6d
82d0 : 0e 26 a2 f0 ab 2c 0f 0c   28
82d8 : 0b 00 00 00 00 00 00 00   e4
82e0 : 00 00 00 00 60 e0 f0 e0   73
82e8 : 60 00 b0 c0 f6 ce bf 0e   8d
82f0 : 26 aa f2 a0 2b 0c 0f 0c   a3
82f8 : 0b 00 00 00 00 00 00 00   04
8300 : 00 00 00 60 e0 f0 e0 60   e6
8308 : 00 b0 c0 f6 ce bf 0e 26   df
8310 : af fa a2 20 0b 0c 0f 0c   4e
8318 : 0b 00 00 00 00 00 00 00   24
8320 : 00 00 60 e0 f0 e0 60 00   ec
8328 : b0 c0 f6 ce bf 0e 26 aa   2a
8330 : ff aa 22 00 0b 0c 0f 0c   72
8338 : 0b 00 00 00 00 00 00 00   44
8340 : 00 60 e0 f0 e0 60 00 b0   39
8348 : c0 f6 ce bf 0e 26 a2 fa   c1
8350 : af 2a 02 00 0b 0c 0f 0c   fa
8358 : 0b 00 00 00 00 00 00 00   64
8360 : 00 00 60 e0 f0 e0 60 b0   8d
8368 : c6 fe cf be 26 a0 f2 aa   01
8370 : 2f 0a 02 00 0b 0c 0f 0c   8a
8378 : 0b 00 00 00 00 00 00 00   84
8380 : 00 00 00 60 e0 f0 e0 b6   13
8388 : ce ff ce 26 a0 f0 a2 2a   3f
8390 : 0f 0a 02 00 0b 0c 0f 0c   8a
8398 : 0b 00 00 00 00 00 00 00   a4
83a0 : 00 00 00 00 60 e0 f6 be   07
83a8 : cf fe 26 a0 f0 a0 22 0a   45
83b0 : 0f 0a 02 00 0b 0c 0f 0c   aa
83b8 : 0b 00 00 00 00 00 00 00   c4
83c0 : 00 00 00 00 00 66 ee bf   2f
83c8 : ce 26 a0 f0 a0 20 02 0a   17
83d0 : 0f 0a 02 00 0b 0c 0f 0c   ca
83d8 : 0b 00 00 00 00 00 00 00   e4
83e0 : 00 00 00 00 06 0e 6f be   ed
83e8 : 26 a0 f0 a0 20 00 02 0a   cd
83f0 : 0f 0a 02 00 0b 0c 0f 0c   ea
83f8 : 0b 00 00 00 00 00 00 00   04
8400 : 00 00 00 06 0e 0f 0e 26   9f
8408 : a0 f0 a0 20 00 00 02 0a   69
8410 : 0f 0a 02 00 0b 0c 0f 0c   0a
8418 : 0b 00 00 00 00 00 00 00   24
8420 : 00 00 06 0e 0f 0e 26 a0   9f
8428 : f0 a0 20 b0 60 00 02 0a   a9
8430 : 0f 0a 02 00 0b 0c 0f 0c   2a
8438 : 0b 00 00 00 00 00 00 00   44
8440 : 00 00 06 0e 0f 2e a6 f0   63
8448 : a0 20 c0 b0 e0 60 02 0a   6c
8450 : 0f 0a 02 0b 0c 0f 0c 0b   c6
8458 : 00 00 00 00 00 00 00 00   59
8460 : 00 00 06 0e 2f ae f6 a0   29
8468 : 20 f0 c0 b0 f0 e0 62 0a   fa
8470 : 0f 0a 0b 0c 0f 0c 0b 00   46
8478 : 00 00 00 00 00 00 00 00   79
8480 : 00 00 06 2e af fe a6 20   95
8488 : c0 f0 c0 b0 e0 f0 e2 6a   fc
8490 : 0f 0b 0c 0f 0c 0b 00 00   23
8498 : 00 00 00 00 00 00 00 00   99
84a0 : 00 00 26 ae ff ae 26 b0   6f
84a8 : c0 f0 c0 b0 60 e0 f2 ea   d5
84b0 : 6b 0c 0f 0c 0b 00 00 00   17
84b8 : 00 00 00 00 00 00 00 00   b9
84c0 : 00 20 a6 fe af 2e 06 b0   40
84c8 : c0 f0 c0 b0 00 60 e2 fb   cd
84d0 : ec 6f 0c 0b 00 00 00 00   d8
84d8 : 00 00 00 00 00 00 00 00   d9
84e0 : 00 00 26 ae ff ae 26 00   4e
84e8 : b0 c0 f0 c0 b0 00 6b ec   df
84f0 : ff ec 6b 00 00 00 00 00   40
84f8 : 00 00 00 00 00 00 00 00   f9
8500 : 00 00 06 2e af fe a6 20   15
8508 : 00 b0 c0 f0 c0 bb 0c 6f   a7
8510 : ec fb e2 60 00 00 00 00   bf
8518 : 00 00 00 00 00 00 00 00   19
8520 : 00 00 06 0e 2f ae f6 a0   e9
8528 : 20 00 b0 c0 fb cc bf 0c   ca
8530 : 6b ea f2 e0 60 00 00 00   ef
8538 : 00 00 00 00 00 00 00 00   39
8540 : 00 00 06 0e 0f 2e a6 f0   63
8548 : a0 20 00 bb cc ff cc bb   e7
8550 : 0f 6a e2 f0 e0 60 00 00   7c
8558 : 00 00 00 00 00 00 00 00   59
8560 : 00 00 06 0e 0f 0e 26 a0   df
8568 : f0 a0 2b 0c bf cc fb ca   dd
8570 : bf 0a 62 e0 f0 e0 60 00   81
8578 : 00 00 00 00 00 00 00 00   79
8580 : 00 00 06 0e 0f 0e 06 20   7d
8588 : a0 fb ac 2f 0c bb c2 fa   d6
8590 : cf ba 02 60 e0 f0 e0 60   23
8598 : 00 00 00 00 00 00 00 00   99
85a0 : 00 00 06 0e 0f 0e 06 00   5d
85a8 : 2b ac ff ac 2b 00 b2 ca   d2
85b0 : ff ca b2 00 60 e0 f0 e0   54
85b8 : 60 00 00 00 00 00 00 00   19
85c0 : 00 00 00 06 0e 0f 0e 06   1f
85c8 : 00 2b ac ff ac 2b 00 b2   12
85d0 : ca ff ca b2 e0 f0 e0 60   7d
85d8 : 00 00 00 00 00 00 00 00   d9
85e0 : 00 00 00 00 06 0e 0f 0e   0a
85e8 : 06 00 2b ac ff ac 2b 00   61
85f0 : b2 ca ff ca b2 e0 60 00   14
85f8 : 00 00 00 00 00 00 00 00   f9
8600 : 00 00 00 00 00 06 0e 0f   87
8608 : 0e 06 00 2b ac ff ac 2b   53
8610 : 00 b2 ca ff ca b2 00 00   5e
8618 : 00 00 00 00 00 00 00 00   19
8620 : 00 00 00 00 00 00 06 0e   55
8628 : 0f 0e 06 00 2b ac ff ac   31
8630 : 2b e0 b2 ca ff ca b2 00   f2
8638 : 00 00 00 00 00 00 00 00   39
8640 : 00 00 00 00 00 00 00 06   4d
8648 : 0e 0f 0e 06 00 2b ac ff   2e
8650 : ac 2b e0 b2 ca ff ca b2   5d
8658 : 00 00 00 00 00 00 00 00   59
8660 : 00 00 00 00 00 00 00 00   61
8668 : 06 0e 0f 0e 06 00 2b ac   61
8670 : ff ac 2b 00 b2 ca ff ca   a7
8678 : b2 00 00 00 00 00 00 00   2b
8680 : 00 00 00 00 00 00 00 06   8d
8688 : 0e 0f 0e 06 00 6b ec 2f   d0
8690 : ac fb a0 22 ca ff ca b2   e3
8698 : 00 00 00 00 00 00 00 00   99
86a0 : 00 00 00 00 00 00 06 0e   d5
86a8 : 0f 0e 06 00 6b ec ff ec   38
86b0 : 2b a0 f2 aa 2f ca b2 00   51
86b8 : 00 00 00 00 00 00 00 00   b9
86c0 : 00 00 00 00 00 06 0e 0f   47
86c8 : 0e 06 00 6b ec ff ec 6b   a0
86d0 : 00 22 aa ff aa 22 00 00   48
86d8 : 00 00 00 00 00 00 00 00   d9
86e0 : 00 00 00 00 06 0e 0f 0e   0a
86e8 : 06 00 6b ec ff ec 6b 00   7c
86f0 : b2 ca 2f aa f2 a0 20 00   dd
86f8 : 00 00 00 00 00 00 00 00   f9
```

Listing 5. Das MSE-Listing eines herrlichen MBB-Films. Es dient als Demonstration der Leistungen von MBB und kann im Editor mit dem »L«-Befehl geladen werden.

64ER ONLINE

```
8700 : 00 00 00 06 0e 0f 0e 06   5f
8708 : 00 6b ec ff ec 6b 00 b2   88
8710 : ca ff ca 22 a0 f0 a0 20   25
8718 : 00 00 00 00 00 00 00 00   19
8720 : 00 00 06 0e 0f 0e 06 00   dd
8728 : 6b ec ff ec 6b 00 b2 ca   be
8730 : ff ca b2 00 20 a0 f0 a0   4d
8738 : 20 00 00 00 00 00 00 00   59
8740 : 00 06 0e 0f 0e 06 00 6b   91
8748 : ec ff ec 6b 00 b2 ca ff   9d
8750 : ca b2 00 20 a0 f0 a0 20   cc
8758 : 00 00 00 00 00 00 00 00   59
8760 : 00 00 06 0e 0f 0e 66 ec   79
8768 : ff ec 6b 00 b2 ca ff ca   cf
8770 : b2 00 20 a0 f0 a0 20 00   d3
8778 : 00 00 00 00 00 00 00 00   79
8780 : 00 00 00 06 0e 6f ee f6   47
8788 : ec 6b 00 b2 ca ff ca b2   bd
8790 : 00 20 a0 f0 a0 20 00 00   f2
8798 : 00 00 00 00 00 00 00 00   99
87a0 : 00 00 00 00 66 ee ff ee   5c
87a8 : 66 00 b2 ca ff ca b2 00   35
87b0 : 20 a0 f0 a0 20 00 00 00   73
87b8 : 00 00 00 00 00 00 00 00   b9
87c0 : 00 00 00 6b ec f6 ee 6f   4f
87c8 : 0e 06 ba cf fa c2 b0 00   0a
87d0 : 00 20 a0 f0 a0 20 00 00   32
87d8 : 00 00 00 00 00 00 00 00   d9
87e0 : 00 00 6b ec ff ec 66 0e   76
87e8 : 0f 0e b6 ca f2 c0 b0 00   fd
87f0 : 00 00 20 a0 f0 a0 20 00   a1
87f8 : 00 00 00 00 00 00 00 00   f9
8800 : 00 6b ec ff ec 6b 00 06   27
8808 : 0e 0f be c6 f0 c0 b0 00   fe
8810 : 00 00 00 20 a0 f0 a0 20   69
8818 : 00 00 00 00 00 00 00 00   19
8820 : 00 00 6b ec ff ec 6b 0a   c2
8828 : 06 0e bf ce f6 c0 b0 00   37
8830 : 00 00 20 a0 f0 a0 20 00   e1
8838 : 00 00 00 00 00 00 00 00   39
8840 : 00 00 00 6b ec ff ec 6b   07
8848 : 0a 06 be cf fe c6 b0 00   e8
8850 : 00 20 a0 f0 a0 20 00 00   b2
8858 : 00 00 00 00 00 00 00 00   59
8860 : 00 00 00 00 6b ec ff ec   58
8868 : 6b 00 b6 ce ff ce b6 00   ac
8870 : 20 a0 f0 a0 20 00 00 00   33
8878 : 00 00 00 00 00 00 00 00   79
8880 : 00 00 00 02 0a 6b ec ff   70
8888 : ec 6b b0 c6 fe cf be 26   e4
8890 : a0 f0 a0 20 00 00 00 00   d5
8898 : 00 00 00 00 00 00 00 00   99
88a0 : 00 00 02 0a 0f 0a 6b ec   2b
88a8 : ff ec bb c0 f6 ce 2f ae   24
88b0 : f6 a0 20 00 00 00 00 00   ff
88b8 : 00 00 00 00 00 00 00 00   b9
88c0 : 00 02 0a 0f 0a 02 00 6b   ad
88c8 : ec ff bc cb f0 26 ae ff   57
88d0 : ae 26 00 00 00 00 00 00   92
88d8 : 00 00 00 00 00 00 00 00   d9
88e0 : 00 00 02 0a 0f 0a 02 00   ec
88e8 : 6b ec bf cc 2b a0 f6 ae   44
88f0 : 2f 0e 06 00 00 00 00 00   a8
88f8 : 00 00 00 00 00 00 00 00   f9
8900 : 00 00 00 02 0a 0f 0a 02   86
8908 : 00 6b bc 2f ac fb a0 26   4c
8910 : 0e 0f 0e 06 00 00 00 00   ea
8918 : 00 00 00 00 00 00 00 00   19
8920 : 00 00 00 00 02 0a 0f 0a   e1
8928 : 02 00 2b ac ff ac 2b 00   9d
8930 : 06 0e 0f 0e 06 00 00 00   23
8938 : 00 00 00 00 00 00 00 00   39
8940 : 00 00 00 00 00 02 0a 0f   97
8948 : 0a 22 a0 fb ac 2f bc 6b   19
8950 : 00 06 0e 0f 0e 06 00 00   ca
8958 : 00 00 00 00 00 00 00 00   59
8960 : 00 00 00 00 00 00 02 0a   7d
8968 : 2f aa f2 a0 2b cc bf ec   af
8970 : 6b 00 06 0e 0f 0e 06 00   98
8978 : 00 00 00 00 00 00 00 00   79
8980 : 00 00 00 00 00 00 00 22   c5
8988 : aa ff aa 22 f0 cb bc ff   81
8990 : ec 6b 00 06 0e 0f 0e 06   90
8998 : 00 00 00 00 00 00 00 00   99
89a0 : 00 00 00 00 00 00 20 a0   62
89a8 : f2 aa 2f ca f2 c0 bb ec   12
89b0 : ff ec 6b 00 06 0e 0f 0e   29
89b8 : 06 00 00 00 00 00 00 00   bf
89c0 : 00 00 00 00 00 20 a0 f0   26
89c8 : a0 22 ba cf fa c2 bc 6f   b9
89d0 : ec fb e0 60 06 0e 0f 0e   27
89d8 : 06 00 00 00 00 00 00 00   df
89e0 : 00 00 00 00 20 a0 f0 a0   ed
89e8 : 20 00 b2 ca ff ca b2 0c   47
89f0 : 6b e0 f0 e0 66 0e 0f 0e   53
89f8 : 06 00 00 00 00 00 00 00   ff
8a00 : 00 00 00 20 a0 f0 a0 20   59
8a08 : 00 00 b0 c2 fa cf ba 02   aa
8a10 : 00 60 e0 f0 e6 6e 0f 0e   d1
8a18 : 06 00 00 00 00 00 00 00   1f
8a20 : 00 00 20 a0 f0 a0 20 00   d1
8a28 : 00 00 bb cc f2 ca bf 0a   49
8a30 : 02 00 60 e0 f6 ee 6f 0e   27
8a38 : 06 00 00 00 00 00 00 00   3f
8a40 : 00 20 a0 f0 a0 20 00 00   a2
8a48 : 00 0b bc cf fc c2 ba 0f   e6
8a50 : 0a 02 00 60 e6 fe ef 6e   6a
8a58 : 06 00 00 00 00 00 00 00   5f
8a60 : 00 00 20 a0 f0 a0 20 00   11
8a68 : 0b 0c bf cc fb c0 b2 0a   a7
8a70 : 0f 0a 62 e0 f6 ee 6f 0e   fa
8a78 : 06 00 00 00 00 00 00 00   7f
8a80 : 00 00 00 20 a0 f0 a0 20   d9
8a88 : 0b 0c bf cc fb c0 b0 02   af
8a90 : 0a 6f ea f6 ee 6f 0e 06   9a
8a98 : 00 00 00 00 00 00 00 00   99
8aa0 : 00 00 00 00 20 a0 f0 a0   ad
8aa8 : 2b 0c bf cc fb c0 b0 00   eb
8ab0 : 62 ea f6 ee 6f 0e 06 00   a2
8ab8 : 00 00 00 00 00 00 00 00   b9
8ac0 : 00 00 00 00 00 20 a0 f0   26
8ac8 : ab 2c bf cc fb c0 b0 60   5c
8ad0 : e0 f6 ee 6f 0e 06 00 00   e6
8ad8 : 00 00 00 00 00 00 00 00   d9
8ae0 : 00 00 00 00 00 00 20 a0   a2
8ae8 : fb ac bf cc fb c0 60 e0   cc
8af0 : f6 ee 6f 0e 06 0a 02 00   b4
8af8 : 00 00 00 00 00 00 00 00   f9
8b00 : 00 00 00 00 00 00 00 20   41
8b08 : ab fc bf cc fb 60 e0 f6   ef
8b10 : ee 6f 0e 06 0a 0f 0a 02   40
8b18 : 00 00 00 00 00 00 00 00   19
8b20 : 00 00 00 00 00 00 00 00   21
8b28 : 2b ac bf cc 6b e0 f6 ee   aa
8b30 : 6f 0e 06 00 02 0a 0f 0a   e9
8b38 : 02 00 00 00 00 00 00 00   3b
8b40 : 00 00 00 00 00 00 00 00   41
8b48 : 00 2b bc 6f ec fb ee 6f   44
8b50 : 0e 06 00 00 02 0a 0f 0a   22
8b58 : 02 00 00 00 00 00 00 00   5b
8b60 : 00 00 00 00 00 00 00 00   61
8b68 : 00 00 6b ec ff ec 6b 0e   12
8b70 : 06 00 00 00 02 0a 0f 0a   37
8b78 : 02 00 00 00 00 00 00 00   7b
8b80 : 00 00 00 00 00 00 00 00   81
8b88 : 00 60 e0 fb ec 6f bc 2b   03
8b90 : 00 00 00 00 02 0a 0f 0a   51
8b98 : 02 00 00 00 00 00 00 00   9b
8ba0 : 00 00 00 00 00 00 00 00   a1
8ba8 : 60 e0 f6 ee 6b cc bf ac   89
8bb0 : 2b 00 00 00 02 0a 0f 0a   9c
8bb8 : 02 00 00 00 00 00 00 00   bb
8bc0 : 00 00 00 00 00 00 00 60   81
8bc8 : e0 f6 ee 6f fe cb bc ff   0e
8bd0 : ac 2b 00 00 02 0a 0f 0a   d3
8bd8 : 02 00 00 00 00 00 00 00   db
8be0 : 00 00 00 00 00 00 60 e0   24
8be8 : f6 ee 6f ce f6 c0 bb ac   c9
8bf0 : ff ac 2b 00 02 0a 0f 0a   d1
8bf8 : 02 00 00 00 00 00 00 00   fb
8c00 : 00 00 00 00 00 60 e0 f0   69
8c08 : e0 66 be cf fe c6 bb 2c   32
8c10 : af fc ab 20 02 0a 0f 0a   ed
8c18 : 02 00 00 00 00 00 00 00   1b
8c20 : 00 00 00 00 60 e0 f0 e0   b3
8c28 : 60 00 b6 ce ff ce b6 0c   79
8c30 : 2f ac fb a0 22 0a 0f 0a   8b
8c38 : 02 00 00 00 00 00 00 00   3b
8c40 : 00 00 00 60 e0 f0 e0 60   26
8c48 : 00 00 b0 c6 fe cf be 06   c3
8c50 : 0f 2c ab f0 a2 2a 0f 0a   4a
8c58 : 02 00 00 00 00 00 00 00   5b
8c60 : 00 00 60 e0 f0 e0 60 00   2c
8c68 : 00 00 b0 c0 f6 ce bf 0e   ad
8c70 : 06 0c 2b a0 f2 aa 2f 0a   b1
8c78 : 02 00 00 00 00 00 00 00   7b
8c80 : 00 60 e0 f0 e0 60 00 00   18
8c88 : 00 00 b0 c0 f0 c6 be 0f   2b
8c90 : 0e 06 0b 20 a2 fa af 2a   7d
8c98 : 02 00 00 00 00 00 00 00   9b
8ca0 : 00 00 60 e0 f0 e0 60 00   6c
8ca8 : 00 00 b0 c0 f0 c0 b6 0e   f8
8cb0 : 0f 0e 26 a0 f2 aa 2f 0a   b9
8cb8 : 02 00 00 00 00 00 00 00   bb
8cc0 : 00 00 00 60 e0 f0 e0 60   a6
8cc8 : 00 00 b0 c0 f0 c0 b0 06   f0
8cd0 : 0e 2f ae f6 a2 2a 0f 0a   cc
8cd8 : 02 00 00 00 00 00 00 00   db
8ce0 : 00 00 00 00 60 e0 f0 e0   73
8ce8 : 60 00 b0 c0 f0 c0 b0 00   64
8cf0 : 26 ae ff ae 26 0a 0f 0a   46
8cf8 : 02 00 00 00 00 00 00 00   fb
8d00 : 00 00 00 60 e0 f0 e0 60   e6
8d08 : 00 b0 c0 f0 c0 b0 00 20   80
8d10 : a0 f6 ae 2f 0e 06 0f 0a   1e
8d18 : 02 00 00 00 00 00 00 00   1b
8d20 : 00 00 60 e0 f0 e0 60 00   ec
8d28 : b0 c0 f0 c0 b0 00 20 a0   59
8d30 : f0 a0 26 0e 0f 0e 06 0a   49
8d38 : 02 00 00 00 00 00 00 00   3b
8d40 : 00 60 e0 f0 e0 60 00 b0   39
8d48 : c0 f0 c0 b0 00 20 a0 f0   2c
8d50 : a0 20 00 06 0e 0f 0e 06   5f
8d58 : 02 00 00 00 00 00 00 00   5b
8d60 : 00 60 e0 f0 e0 60 b0 c0   3c
8d68 : f0 c0 b0 00 20 a0 f0 a0   f1
8d70 : 20 00 00 00 06 0e 0f 0e   ba
8d78 : 06 00 00 00 00 00 00 00   7f
8d80 : 00 60 e0 f0 e0 60 c0 f0   fc
8d88 : c0 b0 00 20 a0 f0 a0 20   f9
8d90 : 00 00 00 06 0e 0f 0e 06   ef
8d98 : 00 00 00 00 00 00 00 00   99
8da0 : 00 60 e0 f0 e0 60 f0 c0   7d
8da8 : b0 00 20 a0 f0 a0 20 00   09
8db0 : 00 00 06 0e 0f 0e 06 00   6d
8db8 : 00 00 00 00 00 00 00 00   b9
8dc0 : 00 60 e0 f0 e0 60 c0 b0   bc
8dc8 : 00 20 a0 f0 a0 20 00 00   2a
8dd0 : 00 06 0e 0f 0e 06 00 00   4a
8dd8 : 00 00 00 00 00 00 00 00   d9
8de0 : 00 60 e0 f0 e0 60 b0 00   3a
8de8 : 20 a0 f0 a0 20 00 00 00   ab
8df0 : 06 0e 0f 0e 06 00 00 00   e3
8df8 : 00 00 00 00 00 00 00 00   f9
8e00 : 00 60 e0 f0 e0 60 00 20   d8
8e08 : a0 f0 a0 20 00 00 00 06   59
8e10 : 0e 0f 0e 06 00 00 00 00   ea
8e18 : 00 00 00 00 00 00 00 00   19
8e20 : 00 60 e0 f0 e0 60 20 a0   79
8e28 : f0 a0 20 00 00 00 06 0e   a5
8e30 : 0f 0e 06 00 00 00 00 00   c8
8e38 : 00 00 00 00 00 00 00 00   39
8e40 : 00 60 e0 f0 e0 60 a0 f0   3c
8e48 : a0 20 00 00 00 06 0e 0f   7f
8e50 : 0e 06 00 00 00 00 00 00   62
8e58 : 00 00 00 00 00 00 00 00   59
8e60 : 00 60 e0 f0 e0 60 f0 a0   fd
8e68 : 20 00 00 00 06 0e 0f 0e   b2
8e70 : 06 00 00 00 00 00 00 00   77
8e78 : 00 00 00 00 00 00 00 00   79
8e80 : 00 60 e0 f0 e0 60 a0 20   da
8e88 : 00 00 00 06 0e 0f 0e 06   e7
8e90 : 00 00 00 00 00 00 00 00   91
8e98 : 00 00 00 00 00 00 00 00   99
8ea0 : 00 60 e0 f0 e0 60 20 00   b8
8ea8 : 00 00 06 0e 0f 0e 06 00   65
8eb0 : 00 00 00 00 00 00 00 00   b1
8eb8 : 00 00 00 00 00 00 00 00   b9
8ec0 : 00 60 e0 f0 e0 60 00 00   58
8ec8 : 00 06 0e 0f 0e 06 00 00   42
8ed0 : 00 00 00 00 00 00 00 00   d1
8ed8 : 00 00 00 00 00 00 00 00   d9
8ee0 : 00 00 60 e0 f0 e0 60 00   ac
8ee8 : 06 0e 0f 0e 06 00 00 00   db
8ef0 : 00 00 00 00 00 00 00 00   f1
8ef8 : 00 00 00 00 00 00 00 00   f9
8f00 : 00 00 00 60 e0 f0 e0 66   f2
8f08 : 0e 0f 0e 06 00 00 00 00   e2
8f10 : 00 00 00 00 00 00 00 00   11
8f18 : 00 00 00 00 00 00 00 00   19
8f20 : 00 00 00 00 60 e0 f6 ee   e7
8f28 : 6f 0e 06 00 00 00 00 00   20
8f30 : 00 00 00 00 00 00 00 00   31
8f38 : 00 00 00 00 00 00 00 00   39
8f40 : 00 00 00 00 00 66 ee ff   2f
8f48 : ee 66 00 00 00 00 00 00   6a
8f50 : 00 00 00 00 00 00 00 00   51
8f58 : 00 00 00 00 00 00 00 00   59
8f60 : 00 00 00 00 06 0e 6f ee   cd
8f68 : f6 e0 60 00 00 00 00 00   e7
8f70 : 00 00 00 00 00 00 00 00   71
8f78 : 00 00 00 00 00 00 00 00   79
8f80 : 00 00 00 06 0e 0f 0e 66   a0
8f88 : e0 f0 e0 60 00 00 00 00   25
8f90 : 00 00 00 00 00 00 00 00   91
8f98 : 00 00 00 00 00 00 00 00   99
8fa0 : 00 00 06 0e 0f 0e 06 00   5d
8fa8 : 60 e0 f0 e0 60 00 00 00   d7
8fb0 : 00 00 00 00 00 00 00 00   b1
8fb8 : 00 00 00 00 00 00 00 00   b9
8fc0 : 00 00 00 00 00 00 00 00   c1
8fc8 : 00 60 e0 f0 e0 60 00 00   60
8fd0 : 00 00 00 00 00 00 00 00   d1
8fd8 : 00 00 00 00 00 00 00 00   d9
8fe0 : 00 00 00 00 00 00 00 00   e1
8fe8 : 00 00 60 e0 f0 e0 60 00   b4
8ff0 : 00 00 00 00 00 00 00 00   f1
8ff8 : 00 00 00 00 00 00 0e 01   33
9000 : 00 00 fe 01 47 49 43 20   ec
```

Listing 5. (Schluß)



Werkzeugkasten für Assembler-Programmierer

In diesem Beitrag liefern wir Ihnen eine umfangreiche Toolbox zum Programmieren in Assembler. Ein- und Ausgabe-Routinen sind darin ebenso enthalten wie sehr genaue Fließkomma- oder Floppy-Routinen

Beim Programmieren in Assembler stellt man schnell fest, daß es ein geradezu universelles Gesetz gibt: Alles dauert länger und ist aufwendiger als man vorher vermutet hat. Das hat einen ganz einfachen Grund: Während in höheren Programmiersprachen Ein- und Ausgabe, Arithmetik, Kontroll- und Datenstrukturen vorhanden sind – als Teil der Sprache oder wie in C die Ein- und Ausgabe als Standard-Bibliotheken – bietet die Assemblersprache sehr wenig von alledem. Also muß man in jedes Programm viele elementare Routinen einbinden, die man sonst in der Laufzeitbibliothek (Runtime-Modul) eines Compilers oder im Basic-Interpreter vorfinden würde.

Fertige Module

Hier soll nun eine Auswahl von Makros und Unterprogrammen vorgestellt werden, die häufig wiederkehrende Aufgaben wie Ein- und Ausgabe, Fließkomma-Arithmetik und Diskettenbedienung erledigen.
Alle Programme und Makros laufen mit Hypra-Ass. Sie können diese aber natürlich nach vorheriger Konvertierung auch mit Giga-Ass (in diesem Sonderheft Seite 116) verwenden.

Die Verwendung solcher Standard-Pakete hat nicht nur den Vorteil, Arbeit zu sparen. Eigene Programme werden auch übersichtlicher, wenn gleiche Funktionen überall gleich aussehen. Natürlich gibt es auch Nachteile: Eine allgemeine Lösung paßt vielleicht nicht immer optimal in den gegebenen Rahmen. Manchmal schleppt man bei Verwendung einer Standardbibliothek auch Programmteile mit sich, die eigentlich nicht benötigt werden oder es könnte im Einzelfall eine effizientere Lösung gefunden werden. Normalerweise überwiegen jedoch die Vorteile, und man kommt auf jeden Fall mit Hilfe einer Standardlösung zu einem lauffähigen Programm. Wenn die Effizienz an Platz und Zeit dann wirklich so gelitten hat, ist anschließend immer noch Zeit, die nicht benötigten Teile der eingebauten Bibliotheken zu löschen sowie spezielle Anpassungen vorzunehmen.

Ein weiterer Nachteil dieser »Baukasten-Methode« tritt leider speziell beim Hypra-Ass auf: Dieser Assembler unterstützt die Verwendung von Bibliotheken in Quelltextform nicht. Manche Assembler werden nämlich mit einer Anweisung wie »INCLUDE FILENAME« oder »LIB FILENAME« dazu veranlaßt, während des Assemblierens den Inhalt einer Diskettendatei an der aktuellen Stelle einzufügen. Der Hypra-Ass kennt eine solche »INCLUDE«-Anweisung nicht, statt dessen gibt es die »APPEND«-Anweisung. Diese Anweisung bewirkt jedoch nur ein Nachladen und Weiterassemblieren, nachdem der Quelltext im Speicher bereits fertig übersetzt ist. Damit eignet sich »APPEND« natürlich nicht, Bibliotheken einzubinden, deren Inhalt bereits während der Assemblierung des Hauptprogramms gebraucht wird.

Aus diesem Grund werden die hier verwendeten Programm-Module vor Eingabe des eigentlichen Programms mit »/M« (MERGE) vom Hypra-Ass Editor aus zusammengefügt. Dabei muß man darauf achten, daß die Reihenfolge der »MERGE«-Befehle stimmt, da der Editor-Befehl »MERGE« die eingelesene Datei immer am Ende des aktuellen Textes anfügt. In Tabelle 1 sehen Sie eine Auflistung der Module.

Die Endung ».HY« bei den Filenamen soll andeuten, daß es sich um Quelltexte im Hypra-Ass-Format handelt. Die Programme sind alle kommentiert. Diese Kommentare kann man bei Platzproblemen im Textspeicher des Hypra-Ass weglassen, da in einem Anwenderprogramm nur noch die Funktion und nicht die Implementierung der Hilfsroutinen und -makros wichtig ist. Für die Leser, die noch nicht wissen, was Makros sind, soll hier noch eine Erklärung gegeben werden. Diejenigen Leser, bei denen eine solche Erklärung schon Gähnanfälle hervorruft, können diesen Abschnitt ja überspringen. Vielleicht finden aber auch Profis im folgenden Abschnitt noch etwas Neues.

Was ist ein Makro?

Der Hypra-Ass bietet, wie auch andere gute Assembler, die Möglichkeit, eine ganze Folge von Befehlen mit einem Namen zu versehen. Eine solche Kennzeichnung wird mit dem Befehl ».MA« eingeleitet und mit ».RT« abgeschlossen. Ein Beispiel wäre etwa:

```
.MA INCA
CLC
ADC #1
.RT
```

Hier wird also die Befehlsfolge »CLC / ADC #1« mit dem Namen »INCA« versehen. Wenn Hypra-Ass während des Übersetzens diese Makrodefinition findet, wird noch kein Befehl erzeugt; statt dessen merkt sich der Assembler, daß unter dem Namen »INCA« ein Makro vereinbart wurde und wo es im Quelltext steht. Im weiteren Programmtext wird nun das Makro mit dem Befehl »... INCA« aufgerufen, das heißt, der Assembler erhält die Anweisung, anstatt dieser Zeile die Befehlsfolge »CLC / ADC #1« zu übersetzen.

Auf diese Weise ist der Benutzer in der Lage, sozusagen eine »Wunsch-Maschine« zu definieren, indem er Befehle definiert, die der Prozessor eigentlich nicht kennt. Diese neuen Befehle müssen natürlich mit Hilfe der bereits vorhandenen definierbar sein. Als vorhandene Befehle gelten hier auch bereits vereinbarte Makros. In diesem Fall ergibt sich also auch eine Hierarchie von Makros.

Dieses Werkzeug wird noch flexibler durch die Möglichkeit, Parameter anzugeben. Die Definition eines Makros mit Parametern sieht beispielsweise so aus:

```
.MA LDAY (ADR)
LDA ADR
LDY ADR+1
.RT
```

Beim Aufruf muß man nun ebenfalls in Klammern angeben, was »ADR« bei diesem speziellen Makroaufruf bedeuten soll:

```
... LDAY (PT)
```

setzt voraus, daß irgendwo ein Label »PT« definiert wurde und erzeugt die Befehle

```
LDA PT
LDY PT+1
```

Es gibt auch Makros, die mehrere Parameter haben, diese sind dann durch Kommata zu trennen. Beispiel:

```
.MA MOVEB (FROM,TO)
LDA FROM
STA TO
.RT
```

Aufruf:

```
... MOVEB(0,MEMMAP)
```

Als Parameter sind numerische Ausdrücke erlaubt. Wenn der Hypra-Ass einen Makro-Aufruf mit Parametern findet, werden diese mit der normalen Ausdruck- und Auswertungsroutine berechnet und den Parameter-Namen aus der Makro-Definition zugeordnet. Das funktioniert ähnlich der Definition von Labels: Während der Expansion des Makros »MOVEB« (das heißt, während der Hypra-Ass Befehle aus dem Rumpf der Makro-Definition übersetzt) kennt der Assembler zusätzliche Symbole mit den Namen »FROM« und »TO«. Dieses Verfahren ist relativ einfach zu programmieren und schnell in der Ausführung, hat jedoch einen Haken: Es ist nicht möglich, etwas anderes als einen numerischen Ausdruck als Makro-Parameter zu übergeben. So wäre es sicher schön, wenn man ein Makro »WRITE« zur Ausgabe einer Zeichenkette wie folgt definieren könnte:

```
.MA WRITE(STRING)   ;DAS GEHT LEIDER NICHT:
LDA #<(AD)
LDY #>(AD)
BNE PRN
AD .TX STRING
.BY 0
PRN JSR PRINTS
.RT
... WRITE(HELLO, WORLD)
```

Dies ist ein Punkt, wo sich teure kommerzielle Assembler vom Hypra-Ass unterscheiden. Übrigens gibt es auch unabhängig von Makroassemblern sogenannte Makro-Prozessoren, die einfach einen Text lesen, darin Definitionen und Aufrufe von Makros erkennen und auswerten. Des weiteren erzeugen sie einen Ausgabetext ohne die Makrodefinitionen und mit ausgeführten Makro-Expandierungen. Ein solcher Makro-Prozessor ist als Pre-Prozessor für einen Compiler einsetzbar (standardmäßig bei C und PL/1 vorgesehen).

Eine andere Anwendung wäre auch die Textverarbeitung mit Makros für Serienbriefe. Oder etwa die Portierung von

Listing	Name	Zeilennummern	Funktion
1	SYSIOLIB.HY	1000 - 1355	Grundlage für alle anderen I/O-Module
2	HEXIOLIB.HY	2000 - 2375	Ein-/Ausgabe von Hexadezimalzahlen
3	DEZIOLIB.HY	3000 - 3655	Ein-/Ausgabe von Dezimalzahlen
4	STRINGIOLIB.HY	4000 - 4390	Ein-/Ausgabe von Zeichenketten
5	SIXTEEN.HY	5000 - 5665	Makrobibliothek für 16-Bit-Befehle
6	DS.HY	6000 - 6060	spezielles Makro für FPBCD.HY
7	DOSCMD.HY	10000 - 10725	Unterprogramm Floppy-Bedienung
8	FILTER1.HY	100 - 300	Beginn allgemeines Konvertierungsprogramm
9	FILTER2.HY	10000 - 11080	Teil 2 des Konvertierprogramms
10	FPBCD.HY	10000 - 12910	Fließkomma-Arithmetik
11	UPN1.HY	100 - 110	Beginn Anwendungsbeispiel für FPBCD.HY
12	UPN2.HY	15000 - 17485	Teil 2 des Anwendungsbeispiels
13	EX1.HY	10000 - 10140	Beispiel zu sysiolib-hy
14	EX2.HY	10000 - 10135	Beispiel zu stringiolib.hy
15	EX3.HY	10000 - 10075	Beispiel zu sixteen.hy

Tabelle 1. Überblick über die einzelnen Module

Programmen. Ein Programm wird zum Beispiel dadurch portabel gemacht, daß es in einer fiktiven, dem Problem gut angepaßten »Assemblersprache« mit nicht zuviel verschiedenen Befehlen geschrieben wird. Um nun dieses Programm auf einem real existierenden Rechner zu implementieren, muß man für jeden Befehl aus der Pseudo-Assemblersprache ein Makro definieren, das auf der Zielmaschine Code erzeugt, der die gewünschte Funktion ausführt. Man kann auch Makros verwenden, um den Befehlssatz eines Asssemblers zu erweitern oder an einen anderen Prozessor anzupassen. So gibt es für den C64-Assembler »ASSI« Makropakete, um Code für den Z80, 6809 und andere Prozessoren zu erzeugen.

Makros nach Wunsch

Schließlich sind auch Makro-Pakete schon verwendet worden, um Compiler auf andere Rechner zu übertragen. Damit ist der kleine Exkurs über Makros beendet; es wird wieder ganz praktisch, und zwar mit einer Beschreibung der einzelnen Module:

SYSIOLIB.HY

Dieses Modul (Listing 1) ist die Grundlage aller weiteren mit Ein- und Ausgabe befaßten Bibliotheken. Zunächst werden die häufig benutzten Einsprünge in die Kernal-Sprungtabelle definiert. Das betrifft die Ein- und Ausgabe von Zeichen über die Standardroutinen (BASIN, BASOUT, GETIN), die Handhabung von Files (SETNAM, SETLFS, OPEN, CLOSE, CHKIN, CHKOUT, CLRCHN, READST) und die Routinen zur direkten Bedienung des seriellen Busses (TALK, TALKSA, LISTEN, LISTENSA, UNTALK, UNLISTEN, IECIN, IECOUT) sowie die Abfrage der RUN/STOP-Taste (STOPEQ). Darauf aufbauend werden dann die Makros »MOPEN«, »MCLOSE« und »MDSTAT« definiert. Hier wird eine auch bei den anderen Modulen eingehaltene Konvention deutlich: Alle Makronamen beginnen mit »M«. Das ist vorteilhaft, weil es häufig verwandte Unterprogramme und Makros gibt - zum Beispiel »OPEN« und »MOPEN« - wobei die Makros nur zum bequemeren Aufruf der Unterprogramme dienen. Das Makro »MOPEN« wird in ähnlicher Form wie der Basic-Befehl »OPEN« aufgerufen:

```
... MOPEN (lfn, dev, sa, filename, fnlaenge)
```

Dabei sind »lfn« (logische Filenummer) und »sa« (Sekundäradresse) Konstanten, »dev« (Gerätenummer), »filename« und »fnlaenge« sind Adressen, an denen die jeweilige Angabe zu finden ist.

Ein-/Ausgabe-Module

Es wäre natürlich angenehmer, wenn man direkt beim Makroaufruf zwischen Konstanten und Adressen unterscheiden oder sogar einen konstanten Filenamen direkt angeben könnte. Dies ist mit dem Hypra-Ass aber nicht möglich, da er nur die Übergabe von numerischen Werten als Makro-Parameter gestattet. Die hier getroffene Festlegung, was als Konstante und was als Adresse zu übergeben ist, stellt natürlich nicht die einzige Möglichkeit dar. Man könnte ebensogut alle numerischen Angaben per Adresse übergeben oder weitere Parameter einführen, die angeben, ob etwa eine Sekundäradresse als Konstante oder als Adresse aufzufassen ist. Das würde jedoch den Aufruf des »MOPEN«-Makros umständlicher und unübersichtlicher machen.

Nachdem ein File mit »MOPEN« eröffnet wurde, kann man mit dem Makro »MDSTAT« die Floppy-Statusmeldung einlesen. Als Parameter muß man zwei Adressen angeben: die Adresse, an der die Gerätenummer zu finden ist, sowie die Adresse eines Puffers, in den die Statusmeldung eingelesen werden soll. Dieser Puffer muß mindestens eine Länge von 33 Byte haben. Das Makro sorgt nur für die Übergabe der Parameter an ein in »SYSIOLIB.HY« definiertes Unterprogramm mit dem Namen »DSTAT«, das die eigentliche Arbeit erledigt. Nach Aufruf dieses Unterprogramms ist das Zero-Flag des Prozessors gesetzt, falls die Statusmeldung mit einer »0« begann. Ein Beispiel zur Benutzung dieser Makros ist in Listing 13 (»EX1.HY«) zu finden.

In »SYSIOLIB.HY« werden schließlich noch »PT« und »CR« definiert. Dabei ist »PT« die Adresse eines Zeigers in der Zero-Page, der von den anderen I/O-Bibliotheken benutzt wird (Adresse: $FB), während »CR« die häufig benutzte Konstante 13 (Carriage Return) bezeichnet.

HEXIOLIB.HY

Das Modul »HEXIOLIB.HY« (Listing 2) enthält immer wieder gebrauchte Unterprogramme zur hexadezimalen Ein- und Ausgabe von Ein- und Zwei-Byte-Werten. Dies sind im einzelnen die Routinen »INPUTHEX« und »PRINTHEX« für Zwei-Byte-Werte sowie »INHEXBYTE« und »PRHEXBYTE« für einzelne Bytes. Wort-Werte werden dabei hier wie auch in allen anderen Programmen im Format A/Y übergeben. Das heißt, im Akkumulator befindet sich das niederwertige und im Y-Register das höherwertige Byte. Nach Aufruf der Eingaberoutinen zeigt das Carry-Flag an, ob ein Fehler aufgetreten ist. Die Makros »MINPUTHEX« und »MPRINTHEX« dienen jeweils zur Ein- oder Ausgabe einer 2-Byte-Zahl von/an eine bestimmte Speicheradresse. Der Aufruf lautet dann:

```
...MINPUTHEX(AD)
```

und

```
...MPRINTHEX(AD)
```

DEZIOLIB.HY

Funktion und Syntax dieses Moduls (Listing 3) sind der »HEXIOLIB.HY« angeglichen, nur arbeitet es mit Dezimalzahlen. Allerdings sind hier nur Routinen und Makros zur Behandlung von 2-Byte-Zahlen vorhanden. Die Namen der Unterprogramme lauten »INPUTDEZ« und »PRINTDEZ«, die Makros heißen »MINPUTDEZ« und »MPRINTDEZ«. Wie vorher wird auch hier ein Fehler durch ein gesetztes Carry-Bit angezeigt. Bei der Ausgabe werden führende Nullen nicht mit angezeigt. Dies könnte man leicht ändern, in diesem Fall würde sich die Routine »PRINTDEZ« noch vereinfachen.

STRINGIOLIB.HY

Dieses Modul (Listing 4) bietet die Unterprogramme »INPUTS«, »PRINTS«, »WRITES«, »CRLF« sowie die Makros »MINPUTS« und »MPRINTS«. Die Adresse der Zeichenkette, die ein- oder ausgegeben werden soll, wird wieder in der Registerkombination A/Y (Akkumulator = Low-Byte, Y-Register = High-Byte) übergeben. »PRINTS« gibt die Zeichenkette ab der Adresse in A/Y aus; das Ende der Zeichenkette muß mit einem Null-Byte markiert sein. »INPUTS« liest eine Zeichenkette in den Speicherbereich ein, dessen Adresse in A/Y übergeben wird; zusätzlich muß noch im X-Register die maximale Anzahl der einzulesenden Zeichen angegeben werden. »INPUTS« legt hinter dem letzten eingelesenen Zeichen ein Null-Byte ab, so daß der Pufferbereich um ein Byte länger sein muß als die maximal erlaubte Eingabe. Bei Rückkehr aus dem Unterprogramm »INPUTS« steht im Y-Register die Anzahl der tatsächlich eingelesenen Zeichen.

Die Routine »WRITES« soll es erlauben, Text direkt »inline« anzugeben, so daß man - ähnlich wie bei einem Basic-PRINT-Befehl - auf einen Blick sieht, was ausgege-

ben wird. Dazu wird der auszugebende Text direkt hinter den Unterprogrammaufruf »JSR WRITES« angeordnet. Der Abschluß des Textes wird wieder mit einem Null-Byte markiert. Eine solche Parameterübergabe ist zwar langsamer als der direkte Aufruf von »PRINTS«, das dürfte normalerweise aber keine Rolle spielen. Ein weiterer Nachteil dieser Methode liegt darin, daß der Object-Code durch das Einstreuen von Text natürlich unübersichtlicher wird. Beide Routinen zur Textausgabe können übrigens beliebig langen Text ausgeben. Die Eingaberoutine »INPUTS« ist jedoch auf 255 Zeichen beschränkt.

Die Routine »CRLF« ist nur eine kleine Ergänzung, sie gibt genau ein RETURN-Zeichen ($0D) aus. Die Routinen »INPUTS« und »PRINTS« können auch mit Hilfe der Makros »MINPUTS« und »MPRINTS« aufgerufen werden. Die Syntax lautet:

```
... MINPUTS(BUFFER,LEN)
... MPRINTS(BUFFER)
```

wobei die Länge als Konstante anzugeben ist. Ein einfaches Anwendungsbeispiel ist im Programm »EX2.HY« (Listing 14) angegeben.

SIXTEEN.HY

Bis auf eine einzige ».BY«-Anweisung werden hier (Listing 5) ausschließlich Makros definiert. Diese Makros sollen die Handhabung von 16-Bit-Werten erleichtern. Es gibt allerdings auch einige 8-Bit-Befehle, die aus Gründen der Systematik mit aufgenommen wurden. Die 16-Bit-Befehle verwenden die Registerkombination A/Y als »16-Bit-Akkumulator«. Ein Beispiel hierzu finden Sie im Programm »EX3.HY« (Listing 15).

16-Bit-Werte einfach verarbeitet

Die einzelnen Makros lauten: »LDAY, LDAYI, MOVEB, MOVEBI, MOVEW, MOVEWI, ADDW, ADDWI, SUBW, SUBWI, CMPW, CMPWI, INCW, DECW, PUSHB, PULLB, PUSHW und PULLW«. Als Parameter ist entweder eine Adresse anzugeben oder – bei den Makros für »IMMEDIATE«-Befehle, zu erkennen am »I« am Namensende – ein direkter Wert. Beispiel: »LDAY ($1234)« lädt den Speicherinhalt aus den Adressen $1234 (low) und $1235 (high) nach A/Y. »LDAYI ($1234)« lädt die Konstante $1234 nach A/Y, Das heißt das niederwertige Byte ($34) wird in den Akku geladen und das höherwertige Byte ($12) in das Y-Register. Im gleichen Sinn gehören die Makros »MOVEB« und »MOVEBI« zusammen sowie »MOVEW« und »MOVEWI«, »ADDW« und »ADDWI« etc. Die »MOVE«-Makros müssen mit zwei Parametern aufgerufen werden. Der erste bezeichnet die Quelle, der zweite das Ziel. »MOVEB« steht für »Move Byte« und »MOVEW« für »Move Word«. Beispiel:

```
... MOVEBI (0, STATUS)
```

speichert eine 0 an der Adresse »STATUS«

```
... MOVEW (PT, OLDPT)
```

rettet einen 2-Byte-Zeiger »PT« an die Adresse »OLDPT/OLDPT+1«. Der Inhalt des Akkus sowie des Y-Registers (bei »MOVEW« und »MOVEWI«) wird durch diese Makros verändert. Die arithmetischen Makros »ADDW, ADDWI, SUBW, SUBWI, CMPW und CMPWI« arbeiten analog zu den normalen arithmetischen Befehlen der 6510-CPU: Der derzeitige Inhalt der Registerkombination A/Y wird mit dem angegebenen Immediate-Operanden (bei »ADDWI, SUBWI und CMPWI«) beziehungsweise mit dem Speicherinhalt an der angegebenen Adresse verknüpft und das Ergebnis in A/Y abgelegt. Zu den Vergleichsbefehlen »CMPW« und »CMPWI« sollte man noch wissen, daß nach dem Vergleich das Zero-Flag sowie das Carry-Flag das Ergebnis korrekt wiedergeben, jedoch nicht das Minus-Flag. Das bedeutet, man kann durch einen anschließenden BNE- oder BEQ-Befehl einen Test auf Gleichheit durchführen oder mit BCC/BCS einen vorzeichenlosen Vergleich anstellen. Die Interpretation des Minus-Flags als Ergebnis eines vorzeichenbehafteten Vergleichs gibt jedoch keine korrekten Ergebnisse. Nach einem vorzeichenlosen Vergleich bedeutet ein gesetztes Carry-Bit »größer oder gleich«, das gelöschte Carry-Bit also »kleiner«. Beispiel:

```
... LDAY (WERT1)
... CMPW (WERT2)
BEQ GLEICH
BCC W1KLEINER
BCS W1GRÖSSER
```

Das Ergebnis dieser Abfrage ist nur korrekt, wenn man »WERT1« und »WERT2« als vorzeichenlose Zahlen (0 bis 65535) auffaßt. Das heißt, es gilt 0 < $8000, da $8000 vorzeichenlos als 32768 betrachtet wird. Bei einer Interpretation als vorzeichenbehaftete Zahl würde man 0 > $8000 festsetzen, da in diesem Fall $8000 als -32768 angesehen würde. Die Makros »INCW« und »DECW« gehören schließlich auch noch zu den arithmetischen Makros. Sie inkrementieren oder dekrementieren einen 16-Bit-Wert im Speicher, wobei der Registerinhalt (A, X, Y) nicht verändert wird. Die Makros »PUSHB, PULLB, PUSHW und PULLW« richten schließlich benutzerdefinierte Stacks ein.

Was hier etwas bombastisch klingt, soll nur bedeuten, daß diese Makros per Programm eine Stapelverwaltung durchführen, im Gegensatz zu dem kleinen Hardware-Stack des C64 im Bereich von $0100 bis $01ff. Als Parameter muß jeweils eine Adresse in der Zero-Page angegeben werden, an der ein 2-Byte-Stackpointer angelegt wird. Der Benutzer ist natürlich selbst für die Initialisierung und Überwachung dieser Stackpointer zuständig. Ein Beispiel finden Sie in Listing 15.

Da der Software-Stack mit Hilfe der indirekt-indizierten Adressierung nachgebildet wird, wird bei Benutzung der »PUSH/PULL«-Befehle ein Zwischenspeicher für das Y-Register benötigt. Dieser Zwischenspeicher ist in der Datei »SIXTEEN.HY« unter dem Namen »SAVEY« definiert.

DOSCMD.HY

Hier (Listing 7) wird ein Unterprogramm definiert, das an passender Stelle in solche Programme eingebunden wird, die mit Disk-Files arbeiten. Es erlaubt dem Benutzer, das Directory einer Diskette zu zeigen, Befehle an das Laufwerk zu schicken (etwa »s0:file« zum Löschen von Dateien), die Statusmeldung des Laufwerkes anzuzeigen und Textfiles auszugeben.

Nützliche DOS-Befehle

Damit werden Programme, die vom Benutzer Filenamen erfragen, ziemlich anwenderfreundlich. Da sowohl Zahlen als auch Zeichenketten ein- und ausgegeben werden, werden die Bibliotheken »SYSIOLIB.HY« (sowieso immer erforderlich), »DEZIOLIB.HY«, »STRINGIOLIB.HY« und »SIXTEEN.HY« benötigt. Um »DOSCMD« auszuprobieren, dient ein kurzer »Programm-Kopf« wie folgt:

```
100 - .BA $C000
110 - JMP DOSCMD
```

Dann werden nacheinander die Module »SYSIOLIB.HY, DEZIOLIB.HY, STRINGIOLIB.HY und DOSCMD.HY« mit »/M« (MERGE) geladen und anschließend die Übersetzung gestartet. Mit SYS 49152 ($C000) überzeugt man sich dann vom Erfolg.

FILTER1.HY, FILTER2.HY

Dies ist das Gerüst eines allgemeinen File-Konvertierprogramms. Es führt in der hier angegebenen

Form allerdings noch keine Konvertierung durch, sondern kopiert nur die Eingabe- auf die Ausgabedatei. Um das Programm mit den erforderlichen Modulen zu verbinden, muß man folgende Files nacheinander in den Hypra-Ass »mergen«: »FILTER1.HY« (Listing 8), »SYSIOLIB.HY« (Listing 1), »STRINGIOLIB.HY« (Listing 4), »SIXTEEN.HY« (Listing 5) und »FILTER2.HY« (Listing 9).

File-Handling leichtgemacht

Im Gegensatz zu den anderen Programmen wird hier der Objectcode nicht bei $C000 abgelegt, sondern im Basic-Bereich ab $0801 und daher beim Assemblieren sofort auf Diskette geschrieben. Im Modul »FILTER1.HY« kann man sehen, wie einfach ein Maschinensprache-Programm mit einem Basic-Kopf zu versehen ist, so daß es wie ein Basic-Programm zu laden (LOAD "filename",8) und mit RUN zu starten ist. Die eigentliche Umcodierung muß in der Routine »CONVERT« geschehen. Die Zeichenübergabe erfolgt im Akku, man könnte also eine tabellengesteuerte Umwandlung einfach mit den Befehlen:

```
CONVERT TAX
        LDA TABLE,X
        RTS
TABLE   .BY 0,1,2    ; 256 Byte Umkodierung
```

erreichen. Allerdings ist dies Gerüst nur für eine Zeichen-für-Zeichen-Umwandlung zu gebrauchen. Wenn eine andere Umwandlung erwünscht ist (etwa PRG-Files in SEQ-Files in Hex-Form zur Übertragung von Programmen in Mailboxen), muß man eben die Schleife ab der Marke »CHARLOOP« so ändern, daß nicht mehr für jedes Eingabezeichen genau einmal »CONVERT« und »PUT« aufgerufen wird, sondern etwa immer 16 Byte gesammelt und dann zusammen als Hex-Zeile konvertiert werden. Dieses Programm dient ja nur als Beispiel, das als Anregung für eigene Anwendungen dienen soll. Eine andere Möglichkeit wäre, die Ein- und Ausgabe nicht nur auf Diskfiles zu erlauben, sondern auch die Gerätenummern einzulesen und so gleich die Übertragung der Dateien auf Drucker, Bildschirm etc. mitzuerledigen. Eine solche allgemeine Umlenkung von Ein- und Ausgabe ist besonders dann von Nutzen, wenn man sich angewöhnt, alle Programme damit zu versehen und auf diese Weise die Kopplung zu ermöglichen.

Universelles Konvertierprogramm

Dieses Konzept ist besonders durch das Betriebssystem UNIX für Großrechner und Workstations bekannt geworden. Dort ist es dank Multitasking sogar möglich, die einzelnen Programme, die zusammenwirken sollen, gleichzeitig auszuführen und auf Zwischendateien zu verzichten. MS-DOS hat die Möglichkeit, mit sogenannten »Pipes« Programme zu verketten, übernommen. Allerdings werden hier mangels der Fähigkeit zum Multitasking vom Betriebssytem temporäre Dateien angelegt.

Eine Besonderheit des Filter-Programms ist noch die gepufferte Ausgabe. Dadurch muß bei Benutzung eines Einzellaufwerkes der Lese- und Schreibkopf nicht ständig zwischen dem Lese- und Schreibfile wechseln. Statt dessen wird der Speicher bis $9fff als Puffer benutzt, in dem die Ausgaben zwischengespeichert werden, bis er entweder voll ist oder die gesamte Eingabe gelesen wurde. Falls die Ausgabe auf den Bildschirm umgelenkt wird, sollte man die Pufferung eventuell unterlassen oder wenigstens den Puffer verkleinern, da sonst die Anzeige immer mit großen Pausen erfolgt.

FPBCD.HY, UPN1.HY, UPN2.HY

Das eigentliche Unterprogrammpaket steckt hier im Modul »FPBCD.HY« (Listing 10) während »UPN1.HY« (Listing 11) und »UPN2.HY« (Listing 12) eine darauf aufbauende Anwendung bilden. Die Bezeichnung »FPBCD« steht hier für »Floating-Point/BCD«. Es handelt sich also um Unterprogramme, um Fließkomma-BCD-Arithmetik zu realisieren. »BCD« bedeutet »Binary Coded Decimal«. Bei Fließkomma-Zahlen wird bekanntlich eine Zahl in der Form

$M*B^E$

dargestellt, wobei M für Mantisse, B für Basis und E für Exponent steht. Bei den hier aufgeführten Routinen wird nun als Basis 10 gewählt, die Mantisse in in gepackter BCD-Form gespeichert (2 Dezimalziffern pro Byte) und der Exponent als vorzeichenbehaftete 15-Bit-Binärzahl gespeichert. Im sechzehnten Bit wird dann das Vorzeichen der Mantisse (und damit der gesamten Zahl) untergebracht, die maximale Mantissenlänge beträgt 254 Dezimalstellen.

Durch Veränderung einer Konstanten im Quelltext kann man allerdings die maximale Mantissenlänge auch geringer wählen. Dies reduziert den Platzbedarf der Puffer für Argumente und Ergebnis. Der Vorteil der hier präsentierten Routinen gegenüber der Verwendung der Basic-Fließkommaroutinen liegt zum einen in der Unabhängigkeit vom Basic-Interpreter, dann in dem größeren Wertebereich (Exponent von -16384 bis +16383), der wählbaren Mantissenlänge und der genauen Darstellung von Zahlen, die sich im binären Fließkomma-Format nicht exakt darstellen lassen.

Der letzte Punkt bedarf vielleicht noch einer genaueren Erläuterung: Man hört oft, BCD-Arithmetik sei prinzipiell genauer als Binär-Arithmetik. Das trifft so zunächst natürlich nicht für ganze Zahlen zu, denn diese können, solange der Wertebereich ausreicht, immer exakt berechnet werden. Bei Fließkomma-Zahlen sieht die Sache etwas anders aus. Angenommen, man möchte einen Bruch »1/M« als Fließkommazahl darstellen, so muß man ihn ja in die Darstellung

$1/M = Z_1/B + Z_2/B^2 + Z_3/B^3 +...+ Z_n/B^n$

bringen. Dabei steht »B« für die Basis der Fließkommadarstellung (bei dem hier dargestellten Paket also 10 und bei dem Basic-Fließkommaformat 2), »n« für die Mantissenlänge und »Z_1 bis Z_n« für die einzelnen Ziffern der Mantisse. Nach kurzer Umformung erhält man dann die Gleichung

$B^n / M = Z_1*B^{n-1} + Z_2*B^{n-2} +...+ Z_n$

Diese Gleichung kann nur erfüllt werden, wenn »B^n« durch »M« teilbar ist, also wenn alle Primfaktoren von »M« auch in »B^n« enthalten sind. Dabei wird klar, daß die Wahl von 2 als Exponent nicht sehr günstig ist, denn es lassen sich so nur Brüche »1/M« exakt darstellen, bei denen »M« eine Zweier-Potenz ist. Im Gegensatz dazu enthält 10 die Primfaktoren 2 und 5, so daß sich bei gegebener Mantissenlänge »n« mehr Zahlen genau darstellen lassen, nämlich alle Brüche »1/M«, bei denen sich m in nicht mehr als n Faktoren 2 und 5 zerlegen läßt. Dieser Vergleich ist allerdings in einer Hinsicht nicht gerade fair: Es wurde von gleicher Mantissenlänge n ausgegangen, wobei aber eine Mantisse von n binären Ziffern eben auch nur n Bit benötigt, eine Mantisse von n dezimalen Ziffern bei der gepackten BCD-Darstellung 4 mal n Bit braucht.

Der Vergleich fällt also tatsächlich nicht ganz so schlecht für die binäre Darstellung aus. So kann man etwa bei 4 Byte Mantissenlänge (also 32 binäre oder 8 dezimale Stellen) die Zahl 511/512 = 0.998046875 (dezimal) = 0.111111111 (binär) in der BCD-Darstellung schon nicht mehr exakt wiedergeben, binär jedoch ohne Schwierigkeiten. Die Aussage, BCD-Arithmetik sei immer genauer als Binär-Arithmetik ist also in dieser allgemeinen Formulierung

nicht richtig. Allerdings hat die BCD-Darstellung noch einen großen Vorteil: Da alle Menschen dezimal zu rechnen gewohnt sind, treten die Rundungsfehler bei der BCD-Rechnung genau da auf, wo man sie auch erwartet. So hat sich sicher noch nie jemand gewundert, daß bei der Berechnung von 1/3 ein Fehler auftritt, während es schon verblüffend ist, daß eine Basic-Schleife mit Schrittweite 1/10 von –1 bis 1 die 0 hervorragend verfehlt. Ein weiterer Nachteil der gewählten BCD-Darstellung ist noch der höhere Zeitbedarf, der mit steigender Mantissenlänge quadratisch (bei Multiplikation und Division) zunimmt.

Während bei 8- bis 10stelligen Zahlen die Rechenzeit zwar auch langsamer als bei Basic-Fließkomma-Arithmetik, dabei jedoch noch ganz erträglich ist, dauert eine Division in 254stelliger Genauigkeit für einen Computer geradezu erschreckend lang. Andererseits würde sie mit Papier und Bleistift sicher noch länger dauern...

Genauigkeit durch 254 Dezimalstellen

Die einzelnen von einem Anwenderprogramm verwendbaren Routinen des Moduls »FPBCD.HY« sind »FPADD, FPSUB, FPMULT, FPDIV, LOADARG1, LOADARG2, STOREAKK« und »SETPRECISION«. Es werden immer die beiden Operanden in den internen Puffern »FPARG1« und »FPARG2« erwartet und das Ergebnis in »FPAKK« abgelegt. Zum Belegen der Argumentpuffer dienen die Routinen »LOADARG1« und »LOADARG2«. Sie erhalten im Registerpaar A/Y einen Zeiger auf die im Speicher abgelegte Fließkommazahl, die zu laden ist.

Das Ergebnis kann mit »STOREAKK« dann an eine beliebige Stelle im Speicher (Adresse wieder in A/Y) kopiert werden. Bei der Subtraktion befindet sich der Subtrahend und bei der Division der Divisor im Argumentpuffer 2. Nach Ausführung einer Rechenoperation wird das Overflow-Bit bei Überlauf gesetzt, so daß man immer mit dem Befehl »BVS« zur Fehlerbehandlung springen kann. Bei Unterlauf (Exponent wird kleiner als –16384) wird das Ergebnis ohne Fehlermeldung auf 0 gesetzt.

Vor der ersten Benutzung des Fließkommapakets muß man mit »SETPRECISION« die gewünschte Genauigkeit einstellen, dabei wird im Prozessor-Akkumulator die Mantissenlänge in Byte (also die halbe Ziffernzahl) übergeben. Natürlich darf diese Zahl nicht größer als die Konstante »MAXLEN« im Quelltext sein. »MAXLEN« selbst darf mit maximal 127 vereinbart werden. Das Format einer Fließkommazahl lautet:

(EXP-LOW) (EXP-HIGH) (MANTISSE-1) (MANTISSE-2)...

Dabei wird immer einer normalisierte Darstellung der Form:

$0,Z_1\ Z_2\ Z_3 \ldots . 10^e$

vorausgesetzt. Das heißt, die erste Stelle der Mantisse muß ungleich Null sein und ist als erste Nachkommastelle der Normaldarstellung zu verstehen. Das Vorzeichen der Fließkomma-Zahl ist in Bit 7 des »EXP-HIGH - Byte« untergebracht, der restliche Exponent ist als vorzeichenbehaftete 15-Bit-Zahl zu interpretieren. Einige Beispiele mit der Mantissenlänge 5 Byte sollten das Format klarstellen:

```
PI          .WO 1                       ; PI ist etwa
                                          0.31415926*10↑1
            .BY $31,$41,$59,$26,$54
MILLION     .WO 7                       ; 1 Million
                                          = 10↑6 = 0.1 *
                                          10↑7
            .BY $10,$00,$00,$00,$00
MIMILLION   .WO $8000+7                 ; -1 Million
                                          (Vorzeichen Bit 15)
            .BY $10,$00,$00,$00,$00
HUNDERTSTEL .WO $7FFF                   ; Exponent ist -1,
                                          Vorzeichen der
            .BY $10,$00,$00,$00,$00     ; Mantisse ist 0
TAUSENDSTEL .WO $7FFE                   ; Exponent ist -2,
                                          Vorzeichen der
            .BY $10,$00,$00,$00,$00     ; Mantisse ist 0
MITAUSENDST .WO $FFFE                   ; Exponent ist -2,
                                          Vorzeichen der
            .BY $10,$00,$00,$00,$00     ; Mantisse ist 1
```

Die Routinen des Moduls »FPBCD.HY« benötigen übrigens die Makrobibliothek »SIXTEEN.HY«. Bei der Programmierung des Fließkommapakets zeigte sich ein Problem des Hypra-Ass. Dieser kennt nämlich keinen Befehl, um eine bestimmte Menge Platz zu reservieren, wie dies etwa andere Assembler mit der Anweisung DS 10 (wie Giga-Ass) oder

```
* = * +10
```

können. Solange man nur in den Speicher assembliert, läßt sich der gewünschte Effekt mit

```
PC       .BA PC+10
```

erreichen. Bei der Ausgabe des Code auf eine Datei funktioniert das jedoch nicht. Leider wird aber eine solche Platzreservierung benötigt, um die Fließkommapuffer jeweils entsprechend dem Wert von »MAXLEN« vereinbaren zu können. Aus diesem Grund wird ein Makro »DS« (Listing 6) definiert, das mit einer Assemblerschleife solange ».BY« -Befehle erzeugt, bis der gewünschte Platz reserviert ist. Dieses Verfahren ist aber nicht besonders schnell, man merkt das an der unterschiedlichen Dauer einer Assemblierung für verschiedene Werte von »MAXLEN«.

Die Module »UPN1.HY« und »UPN2.HY« sollen ein Anwendungsbeispiel für die Fließkommaroutinen darstellen. Hier wird ein Tischrechner mit UPN-Bedienung realisiert. »UPN« steht für »umgekehrte polnische Notation«. Das ist die Methode, mit der auch Hewlett-Packard-Taschenrechner arbeiten. Dieses Verfahren benutzt einen Stack von Zahlen. Man kann nun beliebig viele Zahlen eingeben (bis der Stack voll ist) oder einen der Befehle »+, –, * und /«. Bei Eingabe eines dieser Befehle wird die entsprechende Operation mit den obersten beiden Zahlen auf dem Stack durchgeführt, diese vom Stack entfernt und das Ergebnis wieder auf den Stack gelegt.

Rechnen mit »umgekehrt polnischer Notation«

Der hier vorgestellte UPN-Rechner kann noch etwas mehr: Er zeigt immer vor dem Eingabeprompt die Anzahl der auf dem Stack liegenden Zahlen an und kennt noch die zusätzlichen Befehle »D« (DROP, entfernt oberste Zahl vom Stack), »↑« (verdoppelt oberste Zahl), »←« (vertauscht die beiden obersten Zahlen), »?« (Ausgabe der obersten Zahl, diese bleibt auf dem Stack liegen, wird automatisch nach jeder Rechenoperation durchgeführt) und »X« (EXIT).

Bei der Eingabe ist zu beachten, daß die Befehlszeichen immer ohne führende Leerzeichen eingegeben werden müssen und umgekehrt bei der Eingabe von negativen Zahlen wenigstens ein Leerzeichen vor dem Minus-Zeichen stehen muß, da das Minus sonst als Operator fehlinterpretiert wird. Die Eingabeformel lautet:

Vorzeichen Vorkommaanteil ».« Nachkommaanteil »e« Exponentenvorzeichen Exponent

Jedes dieser Teile kann auch entfallen. Vor der Zahl dürfen beliebig viele Leerzeichen stehen, zwischen den einzelnen Zeilen jedoch nicht. Die Ausgabe erfolgt im normalisierten Format, das heißt in der Form:

0.dddddddddddevp oder
-0.dddddddddddevp

wobei »d« für Dezimalziffer, »v« für das Vorzeichen des Exponenten sowie »p« für diesen Exponenten selbst stehen. Ein Beispiel für die Bedienung des UPN-Rechners:

Mit SYS 12 * 4096 starten

00> 2	»00>« ist die Eingabeaufforderung (Prompt)
01>3	2 und 3 eingeben
02>/	Division
01> 0.66666666666667e+0	Ergebnis wird gerundet
01> -20	Leerzeichen ist nötig
02>/	
01>-0.33333333333334e-1	
01>1e1000	
02>/	
01>-0.33333333333334e-1001	
01>x	Ende

Um den UPN-Rechner zu übersetzen, müssen nacheinander folgende Module mit »/M« zusammengefügt werden: »UPN1.HY« (Listing 12), »SYSIOLIB.HY« (Listing 1), »STRINGIOLIB.HY« (Listing 4), »SIXTEEN.HY« (Listing 5), »DS.HY« (Listing 6), »FPBCD.HY« (Listing 10) und »UPN2.HY« (Listing 12). Der Objectcode wird im Bereich ab $C000 abgelegt. Der restliche Platz bis $cfff wird als Stack für die Fließkommazahlen benutzt. Je nachdem, wie groß »MAXLEN« gewählt wurde, passen unterschiedlich viele Zahlen auf den Stack. Ein Überlauf wird vom Programm erkannt.

Zum Schluß noch ein Hinweis: Wenn ein nicht definiertes Makro aufgerufen wird, gibt der Hypra-Ass eine Fehlermeldung mit unsinniger Zeilennummer an. Man findet jedoch die Fehlerzeile relativ einfach, wenn man sich mit dem Befehl »/!« die unsortierte Liste der Labels ausgeben läßt. Diese werden nämlich in der Reihenfolge ausgegeben, in der sie definiert wurden. Man braucht also erst ab der Zeile im Quelltext zu suchen, in der das letzte ausgegebene Label definiert wurde. Alle hier abgedruckten Listings sind mit dem Hypra-Ass, der im Assembler-Sonderheft 8/85 veröffentlicht wurde, einzugeben. Eine Konvertierung ins Giga-Ass-Format ist aber mit dem in diesem Sonderheft veröffentlichten Programm »HYPRA-KONVERT« (Seite 116ff) ebenso möglich. (D. Zabel/Martin Jobst)

Literaturverweise:

(1) Brown, Peter J., Macro Processors (John Wiley & Sons 1974)

(2) Kernighan/Plauger, Programmierwerkzeuge (Springer 1980)

(3) Nabereit, Herbert: Makro-48 – Assembler im Assembler (C't 8/84, S. 44)

(4) Richards, M., The Portability of the BCPL Compiler (Software, Practice and Eperience 1, 1977, S.135)

(5) Schinke, Ulrich: Arithmetik-Unterricht für 6502 und Z80 (c't 3/84, 4/84, 5/84)

(6) van der Wateren, Frits: GPM for the M6800 (Dr. Dobb's Journal, 10/77, S. 22)

```
1000 -;
1005 -;  system i/o library fuer c 64
1010 -;
1015 -; kernal-jumptabelle
1020 -. gl basout=$ffd2          ; zeichen-ausgabe
1025 -. gl basin=$ffcf           ; zeichen-eingabe
1030 -. gl getin=$ffe4           ; wie basic-get
1035 -. gl chkin=$ffc6           ; eingabe auf file umschalten
1040 -. gl chkout=$ffc9          ; ausgabe auf file umschalten
1045 -. gl clrchn=$ffcc          ; ein/ausgabe zurueck auf tastatur und bildschirm
1050 -. gl open=$ffc0            ; file oeffnen (vorher setnam und setlfs)
1055 -. gl close=$ffc3           ; file schliessen (filenr im akku)
1060 -. gl talk=$ffb4            ; geraet am bus als talker adressieren
1065 -. gl untalk=$ffab          ; talker entadressieren
1070 -. gl listen=$ffb1          ; geraet am bus als listener adressieren
1075 -. gl unlisten=$ffae        ; alle listener entadressieren
1080 -. gl iecin=$ffa5           ; direkte eingabe vom bus
1085 -. gl iecout=$ffa8          ; direkte ausgabe auf bus
1090 -. gl talksa=$ff96          ; sekundaeradresse fuer talker senden
1095 -. gl listensa=$ff93        ; sekundaeradresse fuer listener senden
1100 -. gl stopeq=$ffe1          ; stoptaste abfragen (zero-flag gesetzt bei stop)
1105 -. gl setnam=$ffbd          ; adresse und laenge des filenamen uebergeben
1110 -. gl setlfs=$ffba          ; filenummer, geraet und sekundaeradresse ueberg.
1115 -. gl readst=$ffb7          ; status-byte holen (enstspricht st in basic)
1120 -. gl pt=$fb                ; zero-page-zeiger fuer diverse zwecke
1125 -. gl cr=$0d                ; haeufig benutzte konstante
1130 -. ma mopen(fn, dv, sa, name, nlen)
1135 -           lda #fn
1140 -           ldx dv
1145 -           ldy #sa
1150 -           jsr setlfs
1155 -           lda nlen
1160 -           ldx #<(name)
1165 -           ldy #>(name)
1170 -           jsr setnam
1175 -           jsr open
1180 -. rt
1185 -. ma mclose(fn)
1190 -           lda #fn
1195 -           jsr close
1200 -. rt
1205 -. ma mdstat(dv, buffer)
1210 -           lda dv
1215 -           ldx #<(buffer)
1220 -           ldy #>(buffer)
1225 -           jsr dstat
1230 -. rt
1235 -; trick-makro, damit auch in leeren zeilen label gesetzt werden koennen
1240 [illegible] nix
1245 [illegible]
1250 -; holt floppy-statusmeldung in puffer bei x/y, akku=geraetenummer
1255 -; ergebnis: eq wenn ok, ne wenn nicht ok
1260 -dstat      stx pt
1265 -. gl dstat=dstat
1270 -           sty pt+1
1275 -           jsr talk
1280 -           lda #15! o! $60
1285 -           jsr talksa
1290 -           ldy #0
1295 -getstat    jsr iecin
1300 -           sta (pt),y
1305 -           iny
1310 -           cmp #cr
1315 -           bne getstat
1320 -           dey
1325 -           lda #0
1330 -           sta (pt),y
1335 -           jsr untalk
1340 -           ldy #0
1345 -           lda (pt),y
1350 -           and #$0f
1355 -           rts
```

Listing 1. SYSIOLIB.HY. Bitte dieses und alle folgenden Listings mit dem Hypra-Ass aus Sonderheft 8/85 eingeben (oder entsprechend abgeändert mit dem Giga-Ass auf Seite 116)

```
2000 -;
2005 -; unterprogramme und makros zur ein/ausgabe von hexadezimalzahlen
2010 -; unterprogramme:
2015 -; generell: nach eingabe wird das carry-bit gesetzt,
2020 -;           wenn ein fehler festgestellt wurde
2025 -; inhexbyte: einlesen einer 2-stelligen hexadezimalzahl, ergebnis im akku
2030 -; inputhex:  einlesen einer 4-stelligen hexadezimalzahl,
2035 -;            ergebnis im akku (low byte) und im y-register (high-byte)
2040 -; prhexbyte: ausgabe des akku-inhaltes als 2-stellige hexadezimalzahl
2045 -; printhex:  ausgabe von a/y (akku: low, y: high) als 4-stellige hex-zahl
2050 -; makros:
2055 -; minputhex(ad): 4-stellige hexadezimalzahl nach ad/ad+1 einlesen,
2060 -; mprinthex(ad): 2 byte bei ad/ad+1 als 4-stellige hexadezimalzahl ausg.
2065 -inputhex  jsr inhexbyte
2070 -. gl inputhex=inputhex; auch fuer makros bekanntgeben
2075 -          bcs hxerr
2080 -          tay
2085 -inhexbyte jsr inpnib
2090 -          bcs hxerr
2095 -          asl
2100 -          asl
2105 -          asl
2110 -          asl
2115 -          sta pt        ; pt als zwischenspeicher
2120 -          jsr inpnib
2125 -          bcs hxerr
2130 -          ora pt
2135 -hxerr     rts
2140 -inpnib    jsr basin
2145 -          cmp #"0"
2150 -          bcc inperr
2155 -          cmp #$3a      ;"9"+1
2160 -          bcc inpdig
2165 -          cmp #$47      ;"f"+1
2170 -          bcs inperr
2175 -          cmp #"a"
2180 -          bcc inperr
2185 -          sbc #7
2190 -inpdig    and #$0f
2195 -          clc
2200 -          rts
2205 -inperr    sec
2210 -          rts
2215 -printhex  pha
2220 -. gl printhex=printhex
2225 -          pha
2230 -          tya
2235 -          jsr prhexbyte
2240 -          pla
2245 -prhexbyte pha
2250 -          lsr
2255 -          lsr
2260 -          lsr
2265 -          lsr
2270 -          jsr prnib
2275 -          pla
2280 -prnib     and #$0f
2285 -          cmp #10
2290 -          bcc prdig
2295 -          adc #6
2300 -prdig     adc #"0"      ; carry ist geloescht
```

```
2305 -           jmp basout
2310 -.ma minputhex(ad)
2315 -again      jsr inputhex
2320 -           bcc ok
2325 -           lda #"?"
2330 -           jsr basout
2335 -           jmp again
2340 -ok         sta ad
2345 -           sty ad+1
2350 -.rt
2355 -.ma mprinthex(ad)
2360 -           lda ad
2365 -           ldy ad+1
2370 -           jsr printhex
2375 -.rt
```

Listing 2. HEXIOLIB.HY

```
3000 -; ein/ausgabe von 16-bit-werten als
3005 -; dezimalzahlen (ohne vorzeichen)
3010 -; unterprogramme:
3015 -; printdez: a/y als dezimalzahl mit vornullenunterdrueckung ausgeben
3020 -; inputdez: dezimalzahl einlesen, ergebnis in a/y
3025 -;           carry gesetzt bedeutet fehler
3030 -; makros:
3035 -; mprintdez(ad) 16-bit wert an adresse ad/ad+1 dezimal ausgeben
3040 -; minputdez(ad) 16-bit wert einlesen und im speicher bei ad/ad+1 ablegen
3045 -printdez  sta pt
3050 -.gl printdez=printdez; auch in makros bekanntgeben
3055 -          sty pt+1
3060 -          lda #0
3065 -          ldy #4
3070 -clrbuf    sta dezbuffer,y
3075 -          dey
3080 -          bpl clrbuf
3085 -          ldx #16
3090 -          stx dezcount
3095 -hextodec  ldx #5
3100 -          ldy #0
3105 -          clc
3110 -double    lda dezbuffer,y
3115 -          rol
3120 -          cmp #10
3125 -          bcc nocarry
3130 -          sbc #10
3135 -nocarry   sta dezbuffer,y
3140 -          iny
3145 -          dex
3150 -          bne double
3155 -          asl pt
3160 -          rol pt+1
3165 -          bcc null
3170 -incdec    lda dezbuffer,x
3175 -          clc
3180 -          adc #1
3185 -          cmp #10
3190 -          bcc incdecex
3195 -          lda #0
3200 -          sta dezbuffer,x
3205 -          inx
3210 -          bne incdec
3215 -incdecex  sta dezbuffer,x
3220 -null      dec dezcount
3225 -          bne hextodec
3230 -          ldx #4
3235 -decout    lda dezbuffer,x
3240 -          bne not0
3245 -          bit dezcount
3250 -          bpl skiplz      ; fuehrende nullen unterdruecken
3255 -          bmi prdigit
3260 -not0      dec dezcount    ; flag: schon ziffern ausgegeben
3265 -prdigit   ora #"0"
3270 -          jsr basout
3275 -skiplz    dex
3280 -          bpl decout
3285 -          bit dezcount
3290 -          bmi dezoutexit
3295 -          lda #"0"        ; wenn alle nullen unterdrueckt,
3300 -          jsr basout      ; wenigstens eine ausgeben
3305 -dezoutexit rts
3310 -inputdez  ldy #0
3315 -.gl inputdez=inputdez
3320 -          sty pt
3325 -          sty pt+1
3330 -          sty digitflag
3335 -numloop   jsr basin
3340 -          bit digitflag
3345 -          bmi notfirstd
3350 -          cmp #" "        ; leerzeichen ueberlesen
3355 -          beq numloop
3360 -notfirstd cmp #"0"
3365 -          bcc dezinexit
3370 -          sbc #"0"
3375 -          cmp #10
3380 -          bcs dezinexit
3385 -          tax
3390 -          lda #$ff
3395 -          sta digitflag
3400 -          asl pt
3405 -          rol pt+1
3410 -          bcs numerr
3415 -          lda pt+1
3420 -          sta dezcount
3425 -          lda pt
3430 -          asl pt
3435 -          rol pt+1
3440 -          bcs numerr
3445 -          asl pt
3450 -          rol pt+1
3455 -          bcs numerr
3460 -          adc pt
3465 -          sta pt
3470 -          lda pt+1
3475 -          adc dezcount
3480 -          sta pt+1
3485 -          bcs numerr
3490 -          txa
3495 -          adc pt
3500 -          sta pt
3505 -          bcc num1
3510 -          inc pt+1
3515 -          beq numerr
3520 -num1      iny
3525 -          cpy #6
3530 -          bcc numloop
3535 -numerr    sec
3540 -          rts
3545 -dezinexit bit digitflag
3550 -          bpl numerr
3555 -          clc
3560 -          lda pt
3565 -          ldy pt+1
3570 -          rts
3575 -dezcount  .by 0
3580 -dezbuffer .by 0,0,0,0,0
3585 -digitflag .by 0
3590 -.ma mprintdez(ad)
3595 -          lda ad
3600 -          ldy ad+1
3605 -          jsr printdez
3610 -.rt
3615 -.ma minputdez(ad)
3620 -again     jsr inputdez
3625 -          bcc ok
3630 -          lda #"?"
3635 -          jsr basout
3640 -          jmp again
3645 -ok        sta ad
3650 -          sty ad+1
3655 -.rt
```

Listing 3. DEZIOLIB.HY

```
4000 -;
4005 -; zeichenketten ein und ausgabe
4010 -;
4015 -; unterprogramme:
4020 -; inputs: liest zeichenkette ein
4025 -;         a/y zeigt auf ablageadresse, im x-register steht die
4030 -;         maximale anzahl einzulesender zeichen
4035 -;         nach dem aufruf steht im y-register die anzahl der
4040 -;         tatsaechlich gelesenen zeichen
4045 -;         die eingelesene zeichenkette ist am ende mit 0 markiert,
4050 -;         daher muss an der ablageadresse mindestens 1 byte mehr
4055 -;         platz zur verfuegung stehen als zeichen erwartet werden
4060 -; prints: gibt zeichenkette an der adresse a/y aus, ende muss mit
4065 -;         0 markiert sein
4070 -; writes: gibt ebenfalls zeichenkette aus, diese steht direkt im
4075 -;         anschluss an den unterprogrammaufruf und muss mit 0
4080 -;         abgeschlossen werden
4085 -; crlf:   setzt cursor auf den anfang der naechsten zeile
4090 -;
4095 -; makros:
4100 -; minputs(ad,len)    eingabe
4105 -; mprints(ad)        ausgabe
4110 -inputs    stx maxlens
4115 -.gl inputs=inputs; damit inputs auch von makros aus sichtbar ist
4120 -          sta pt
4125 -          sty pt+1
4130 -          ldy #0
4135 -inputlp   cpy maxlens
4140 -          beq inputsx
4145 -          jsr basin
4150 -          cmp #cr
4155 -          beq inputsx
4160 -          sta (pt),y
4165 -          iny
4170 -          bne inputlp
4175 -inputsx   lda #0
4180 -          sta (pt),y
4185 -          rts
4190 -prints    sta pt
4195 -.gl prints=prints
4200 -          sty pt+1
4205 -          ldy #0
4210 -printlp   lda (pt),y
4215 -          beq printsx
4220 -          jsr basout
4225 -          inc pt
4230 -          bne printlp
4235 -          inc pt+1
4240 -          bne printlp
4245 -          bne printlp
4250 -printsx   rts
4255 -maxlens   .by 0
4260 -writes    pla
4265 -          clc
4270 -          adc #1
4275 -          tax
4280 -          pla
4285 -          adc #0
4290 -          tay
4295 -          txa
4300 -          jsr prints
4305 -          lda pt+1
4310 -          pha
4315 -          lda pt
4320 -          pha
4325 -          rts
4330 -crlf      lda #cr
4335 -          jmp basout
4340 -.ma mprints(adr)
4345 -          lda #<(adr)
4350 -          ldy #>(adr)
4355 -          jsr prints
4360 -.rt
4365 -.ma minputs(adr,len)
4370 -          lda #<(adr)
4375 -          ldy #>(adr)
4380 -          ldx #len
4385 -          jsr inputs
4390 -.rt
```

Listing 4. STRINGIOLIB.HY

```
5000 -;
5005 -;        16-bit-macrolib
5010 -;        fuer gebrauch mit hypra-ass
5015 -;        load a/y
5020 -.ma lday(ad)
5025 -             lda ad
5030 -             ldy ad+1
5035 -.rt
5040 -.ma ldayi(ad)
5045 -             lda #<(ad)
5050 -             ldy #>(ad)
5055 -.rt
5060 -;        store a/y
5065 -.ma stay(ad)
5070 -             sta ad
5075 -             sty ad+1
5080 -.rt
5085 -;        move byte
5090 -.ma moveb(from,to)
5095 -             lda from
5100 -             sta to
5105 -.rt
5110 -.ma movebi(val,to)
5115 -             lda #val
5120 -             sta to
5125 -.rt
5130 -;        move word
5135 -.ma movew(from,to)
5140 -             ... lday(from)
5145 -             ... stay(to)
5150 -.rt
5155 -.ma movewi(val,to)
5160 -             ... ldayi(val)
5165 -             ... stay(to)
5170 -.rt
5175 -;        add word
5180 -.ma addw(ad)
5185 -             clc
5190 -             adc ad
5195 -             pha
5200 -             tya
5205 -             adc ad+1
5210 -             tay
5215 -             pla
5220 -.rt
5225 -.ma addwi(val)
5230 -             clc
5235 -             adc #<(val)
5240 -             pha
5245 -             tya
5250 -             adc #>(val)
5255 -             tay
5260 -             pla
5265 -.rt
5270 -;        sub word
5275 -.ma subw(ad)
5280 -             sec
5285 -             sbc ad
5290 -             pha
5295 -             tya
5300 -             sbc ad+1
5305 -             tay
5310 -             pla
5315 -.rt
5320 -.ma subwi(val)
5325 -             sec
5330 -             sbc #<(val)
5335 -             pha
5340 -             tya
5345 -             sbc #>(val)
5350 -             tay
5355 -             pla
5360 -.rt
5365 -;        compare word unsigned
5370 -.ma cmpw(ad)
5375 -             cpy ad+1
5380 -             bne cmpw1
5385 -             cmp ad
5390 -cmpw1        ... nix
5395 -.rt
5400 -.ma cmpwi(val)
5405 -             cpy #>(val)
5410 -             bne cmpw1
5415 -             cmp #<(val)
5420 -cmpw1        ... nix
5425 -.rt
5430 -;        increment
5435 -.ma incw(ad)
5440 -             inc ad
5445 -             bne incw1
5450 -             inc ad+1
5455 -incw1        ... nix
5460 -.rt
5465 -;        decrement
5470 -.ma decw(ad)
5475 -             pha
5480 -             lda ad
5485 -             bne decw1
5490 -             dec ad+1
5495 -decw1        dec ad
5500 -             pla
5505 -.rt
5510 -;        push a on user stack
5515 -.ma pushb(usp)
5520 -             sty savey
5525 -             ldy #0
5530 -             sta (usp),y
5535 -             ... decw(usp)
5540 -             ldy savey
5545 -.rt
5550 -;        push a/y on user stack
5555 -.ma pushw(usp)
5560 -             pha
5565 -             tya
5570 -             ... pushb(usp)
5575 -             pla
5580 -             ... pushb(usp)
5585 -.rt
5590 -;        pull a ...
5595 -.ma pullb(usp)
5600 -             ... incw(usp)
5605 -             sty savey
5610 -             ldy #0
5615 -             lda (usp),y
5620 -             ldy savey
5625 -.rt
5630 -;        pull a/y
5635 -.ma pullw(usp)
5640 -             ... pullb(usp)
5645 -             tay
5650 -             ... pullb(usp)
```

Listing 5. SIXTEEN.HY

```
6000 -.ma ds( len)
6010 -.if len !>! 0
6020 -.by 0
6030 -.eq len=len-1
6040 -.go 6010
6050 -.ei
6060 -.rt
```

Listing 6. DS.HY

```
10000-doscmd    jsr writes
10005-.tx "geraet:8"
10010-.by 157,0; cursor left
10015-          jsr inputdez
10020-          bcc numok
10025-devbad    rts
10030-numok     cpy #1
10035-          bcs devbad
10040-          cmp #32
10045-          bcs devbad
10050-          sta device
10055-          jsr crlf
10060-          jsr writes
10065-.tx "<return> = status msg"
10070-.by cr
10075-.tx "$        = catalog"
10080-.by cr
10085-.tx "!        = list file"
10090-.by cr
10095-.tx "q        = quit"
10100-.by cr,0
10105-loop      lda #">"
10110-          jsr basout
10115-waitkey   jsr basin      ; promptzeichen
10120-          cmp #">"       ; ignorieren
10125-          beq waitkey
10130-          sta line
10135-          cmp #cr
10140-          bne notempty
10145-          ... mdstat(device,line)
10150-          ... mprints(line)
10155-          jsr crlf
10160-          jmp loop
10165-notempty  ... minputs((line+1),31)
10170-          iny
10175-          sty cmdlen
10180-foundcmd  jsr crlf
10185-          lda line
10190-          cmp #"q"
10195-          bne notquit
10200-          rts
10205-notquit   cmp #"!"
10210-          beq dolist
10215-          jmp notlist
10220-dolist    dec cmdlen
10225-          ... mopen(1,device,2,(line+1),cmdlen)
10230-          ... mdstat(device,line)
10235-          beq fileok
10240-          ... mprints(line)
10245-          jsr crlf
10250-          jmp exitlist
10255-fileok    ... movewi(0,linum)
10260-nextlin   ldx #1
10265-          jsr chkin
10270-          ... minputs(line,80)
10275-          jsr readst
10280-          php
10285-          jsr clrchn
10290-          ... incw(linum)
10295-          ... mprintdez(linum)
10300-          lda #":"
10305-          jsr basout
10310-          ... mprints(line)
10315-          jsr crlf
10320-          plp
10325-          bne exitlist
10330-          jsr testkey
10335-          bne nextlin
10340-exitlist  ... mclose(1)
10345-          jmp loop
10350-notlist   cmp #"$"
10355-          bne notcat
10360-          ... mopen(1,device,0,line,cmdlen)
10365-          ldx #1
10370-          jsr chkin
10375-          jsr basin      ; ladesdresse ueberlesen
10380-          jsr basin
10385-          jsr clrchn
10390-catline   ldx #1
10395-          jsr chkin
10400-          jsr basin
10405-          jsr basin      ; linkadresse
10410-          jsr readst
10415-          bne catexit
10420-          jsr basin
10425-          pha
10430-          jsr basin
10435-          tay
10440-          pla
10445-          jsr printdez
```

```
10450-           lda #" "
10455-           jsr basout
10460-prname     jsr basin
10465-           cmp #0
10470-           beq exitline
10475-           jsr basout
10480-           jmp prname
10485-exitline   jsr crlf
10490-           jsr clrchn
10495-           jsr testkey
10500-           bne catline
10505-catexit    jsr clrchn
10510-           ... mclose(1)
10515-           jmp loop
10520-notcat     lda device
10525-           jsr listen
10530-           lda #15! o! $60
10535-           jsr listensa
10540-           ldy #0
10545-sendcmd    lda line,y
10550-           beq cmdready
10555-           jsr iecout
10560-           iny
10565-           bne sendcmd
10570-cmdready   lda #cr
10575-           jsr iecout
10580-           jsr unlisten
10585-           jmp loop
10590-; ueberprueft, ob stop-taste gedrueckt und
10595-; wartet, wenn leertaste gedrueckt
10600-; ergebnis: "eq", wenn stop, "ne" sonst
10605-.eq stop=3
10610-.eq blank=$20
10615-testkey    jsr getin
10620-           cmp #stop
10625-           beq testx
10630-           cmp #blank
10635-           bne testx
10640-waitnext   jsr getin
10645-           cmp #stop
10650-           beq testx
```

Listing 7. DOSCMD.HY

```
100  -; *********************************************
110  -; *                                           *
120  -; *   allgemeiner start eines assembler-      *
130  -; *   programms, das im basic-bereich         *
140  -; *   liegt und mit 'run' gestartet werden    *
150  -; *   soll.                                   *
160  -; *                                           *
170  -; *********************************************
180  -; objectcode muss auf floppy geschickt werden, damit
190  -; hypra-ass nicht ueberschrieben wird
200  -.eq sys=$9e
210  -.eq ramtop=$37
220  -.ob "@0:filter,p,w"
230  -.ba $801
240  -.wo baslink,1987
250  -.by sys
260  -.tx "(2063)"
270  -.by 0
280  -baslink   .wo 0
290  -begin     jmp start
300  -; hier bibliotheken: sysiolib, stringiolib und sixteen einfuegen
```

Listing 8. FILTER1.HY

```
10000-; *********************************************
10010-; *                                           *
10020-; *  ab hier beginnt das eigentliche programm *
10030-; *                                           *
10040-; *********************************************
10050-srcname    .tx "                  "; 18 byte
10060-srclen     .by 0
10070-destname   .tx "                       "; 23 byte
10080-destlen    .by 0
10090-endpt      .wo 0
10100-stat       .by 0
10110-dev        .by 8
10120-msg        .tx "                  "
10130-.tx "                       "; 40 byte platz fuer status
10140-start      jsr crlf
10150-           jsr writes
10160-.tx "allgemeiner filter"
10170-.by cr
10180-.tx " ... keine konvertierung eingebaut"
10190-.by cr
10200-.tx "eingabe    :"
10210-.by 0
10220-           ... minputs(srcname,18)
10230-           cpy #0
10240-           beq exit
10250-           sty srclen
10260-           jsr crlf
10270-           jsr writes
10280-.tx "ausgabe    :"
10290-.by 0
10300-           ... minputs(destname,19)
10310-           cpy #0
10320-           bne go
10330-exit       rts
10340-go         lda #"s"
10350-           sta destname+1,y
10360-           lda #"w"
10370-           sta destname+3,y
10380-           lda #","
10390-           sta destname,y
10400-           sta destname+2,y
10410-           iny
10420-           iny
10430-           iny
10440-           iny
10450-           sty destlen
10460-           ... mopen(1,dev,8,srcname,srclen)
10470-           ... mdstat(dev,msg)
10480-           bne badstat
10490-           ... mopen(2,dev,9,destname,destlen)
10500-           ... mdstat(dev,msg)
10510-           bne badstat
10520-           jsr emptybuf
10530-           ldx #1
10540-           jsr chkin
10550-charloop   jsr basin
10560-           pha
10570-           jsr readst
10580-           sta stat
10590-           pla
10600-           jsr convert
10610-           jsr put
10620-           jsr stopeq
10630-           beq panic
10640-           lda stat
10650-           beq charloop
10660-           jsr flush
10670-panic      jsr clrchn
10680-           ... mclose(1)
10690-           ... mclose(2)
10700-           rts
10710-badstat    ... mprints(msg)
10720-           jsr crlf
10730-           jmp panic
10740-put        pha
10750-           ... lday(pt)
10760-           ... cmpw(endpt)
10770-           bcc putchar
10780-           jsr flush      ; wenn puffer voll, diesen erst ausgeben
10790-putchar    pla
10800-           ldy #0
10810-           sta (pt),y
10820-           ... incw(pt)
10830-           rts
10840-flush      ... movew(pt,endpt)
10850-           jsr clrchn
10860-           ldx #2
10870-           jsr chkout
10880-           ... movewi(bufferstart,pt)
10890-flushloop  ... lday(pt)
10900-           ... cmpw(endpt)
10910-           bcs flushx
10920-           ldy #0
10930-           lda (pt),y
10940-           jsr basout
10950-           ... incw(pt)
10960-           jsr stopeq
10970-           beq flushx
10980-           jmp flushloop
10990-flushx     jsr emptybuf
11000-           ldx #1
11010-           jsr chkin
11020-           rts
11030-emptybuf   ... movewi(bufferstart,pt)
11040-           ... movew(ramtop,endpt)
11050-           rts
11060-convert    rts            ; hier die eigentliche umcodierung einbauen!
11070-bufferstart .by 0
11080-.en
```

Listing 9. FILTER2.HY

```
10000-; ************************************************************
10005-; *                                                          *
10010-; *   bcd-fliesskomma-arithmetik, mantisse bcd mit fplen bytes *
10015-; *         maximale mantissenlaenge: 254 ziffern            *
10020-; *         exponent binaer von -16384 bis +16383            *
10025-; *        vorzeichen gepackt in bit 15 des exponent         *
10030-; ************************************************************
10035-.gl sp=$22
10040-.gl maxlen=15; maximal 30 stellen mantisse voreingestellt
10045-; ------------------------------------------------------------
10050-; aufrufe von fp-routinen:
10055-; setprecision setzt mantissenlaenge (in byte, ergibt akku*2 ziffern)
10060-; loadarg1: laedt fp-zahl bei (a/y) in argumentbereich 1
10065-; loadarg2: laedt fp-zahl bei (a/y) in argumentbereich 2
10070-; storeakk: speichert ergebnis einer operation nach (a/y)
10075-; fpadd, fpsub, fpmult, fpdiv verknuepfen argumente, overflow-flag
10080-; wird bei fliesskomma-ueberlauf gesetzt
10085-setprecision sta fplen
10090-           asl
10095-           sta dblen
10100-           rts
10105-; fliesskommazahl von (a/y) nach argument-puffer 1 oder 2 kopieren
10110-loadarg1   ldx #0
10115-           beq loadarg
10120-loadarg2   ldx #1
10125-loadarg    sta pt
10130-           sty pt+1
10135-           ldy fplen
10140-           iny            ; mantisse + exponent + vorzeichen kopieren
10145-loadloop   lda (pt),y
10150-           cpx #0
10155-           bne putarg2
10160-           sta exp1,y
10165-           beq nextbyte
10170-putarg2    sta exp2,y
10175-nextbyte   dey
10180-           cpy #$ff
10185-           bne loadloop
10190-           cpx #0
10195-           bne unpack2
```

Listing 10. FPBCD.HY

```
10200-            lda exp1+1
10205-            jsr unpacksign
10210-            sta exp1+1
10215-            sty sign1
10220-            rts
10225-unpack2     lda exp2+1
10230-            jsr unpacksign
10235-            sta exp2+1
10240-            sty sign2
10245-            rts
10250-unpacksign ldy #$00        ; vorzeichen aus bit 7 in y-register
10255-            cmp #$80
10260-            and #$7f        ; exponent-bits ausfiltern
10265-            bcc extexp
10270-            dey
10275-extexp      cmp #$40        ; exponent vorzeichenerweitern
10280-            bcc signpos
10285-            ora #$80
10290-signpos     rts
10295-; fliesskomma-akku nach (a/y) abspeichern
10300-storeakk    sta pt
10305-            sty pt+1
10310-            ldy fplen
10315-            iny             ; 2 byte zusaetzlich (vorzeichen und exponent)
10320-storeloo    lda expakk,y
10325-            sta (pt),y
10330-            dey
10335-            cpy #$ff
10340-            bne storeloo
10345-            rts
10350-fpadd       lda sign1
10355-            sta signakk
10360-            cmp sign2
10365-            beq fpadd1
10370-            jsr dosubtr
10375-            rts
10380-fpadd1      jsr doadd
10385-            rts
10390-fpsub       lda sign1
10395-            sta signakk
10400-            cmp sign2
10405-            bne fpsub1
10410-            jsr dosubtr
10415-            rts
10420-fpsub1      jsr doadd
10425-            rts
10430-doadd       jsr normargs    ; argumente anpassen
10435-            ldx fplen
10440-            clc
10445-            sed
10450-            clc
10455-fpaloop     lda fparg1-1,x ; ergebnis um 1 byte nach rechts verschoben
10460-            adc fparg2-1,x ; ablegen, damit platz fuer ueberlauf
10465-            sta fpakku,x   ; vorhanden ist
10470-            dex
10475-            bne fpaloop
10480-            lda #0
10485-            rol            ; uebertrag aus carry berucksichtigen
10490-            sta fpakku
10495-            cld
10500-            lda expakk
10505-            clc
10510-            adc #2         ; ergebnis ist ja um 1 byte = 2 stellen
10515-            sta expakk
10520-            lda expakk+1   ; nach rechts verschoben, dies beruecksichtigen
10525-            adc #0
10530-            sta expakk+1
10535-            inc fplen
10540-            jsr normakku   ; ergebnis normalisieren
10545-            dec fplen
10550-            bvs fpaovl
10555-            jsr roundakk   ; und runden
10560-            jsr chkmnull
10565-            jsr packsign   ; vorzeichen und exponent packen
10570-fpaovl      rts
10575-dosubtr     jsr compabs
10580-            bcs noxch
10585-            ldx fplen      ; falls abs(fparg1)<abs(fparg2),
10590-xchloop     lda fparg1-1,x ; vor subtraktion vertauschen
10595-            ldy fparg2-1,x
10600-            sta fparg2-1,x
10605-            tya
10610-            sta fparg1-1,x
10615-            dex
10620-            bne xchloop
10625-            ldx #1         ; exponenten austauschen
10630-swapexp     lda exp1,x
10635-            ldy exp2,x
10640-            sta exp2,x
10645-            tya
10650-            sta exp1,x
10655-            dex
10660-            bpl swapexp
10665-            lda signakk
10670-            eor #$80
10675-            sta signakk
10680-noxch       jsr normargs
10685-            ldx fplen
10690-            sed
10695-            sec
10700-fpsloop     lda fparg1-1,x
10705-            sbc fparg2-1,x
10710-            sta fpakku-1,x
10715-            dex
10720-            bne fpsloop
10725-            cld
10730-            jsr normakku
10735-            bvs fpsovl
10740-            jsr roundakk
10745-            jsr chkmnull
10750-            jsr packsign
10755-fpsovl      rts
10760-; packt mantissenvorzeichen und exponent in 2 byte (bei expakk/expakk+1)
10765-; setzt overflow-bit, wenn exponent nicht im bereich von
10770-; -16384 bis 16383 liegt
10775-packsign    lda expakk+1
10780-            and #$c0       ; %11000000
10785-            beq expakkok
10790-            cmp #$c0
10795-            beq expakkok
10800-            cmp #$80
10805-            beq toosmall
10810-            bit overflow
10815-            rts
10820-expakkok    lda expakk+1
10825-            and #$7f
10830-            bit signakk
10835-            bpl akkpos
10840-            ora #$80
10845-akkpos      sta expakk+1
10850-            clv
10855-            rts
10860-toosmall    jsr zerofak
10865-            clv
10870-            rts
10875-; ueberpruefe, ob das ergebnis "-0" lautet und korrigiere zu "+0"
              wenn noetig
10880-chkmnull    lda signakk
10885-            bpl nominusn
10890-            lda expakk
10895-            ora expakk+1
10900-            bne nominusn
10905-            lda fpakku
10910-            bne nominusn
10915-            sta signakk
10920-nominusn    rts
10925-; verschiebe mantisse des kleineren arguments so nach rechts, dass
10930-; anschliessend stellenrichtige addition/subtraktion ausgefuehrt
              werden kann
10935-; ausserdem wird der exponent des ergebnisses hier bestimmt
10940-normargs    sec
10945-            lda exp1
10950-            sta expakk
10955-            sbc exp2
10960-            tay
10965-            lda exp1+1
10970-            sta expakk+1
10975-            sbc exp2+1
10980-            bne highdiff
10985-            cpy #0
10990-            beq normargsx
10995-            bne arg2less
11000-; high-byte ist <> 0
11005-highdiff    bpl arg2less
11010-; arg1<arg2, also diff:=-diff und zeiger auf arg1
11015-            pha
11020-            tya
11025-            eor #$ff
11030-            clc
11035-            adc #1
11040-            tay
11045-            pla
11050-            eor #$ff
11055-            adc #0         ; vorzeichenwechsel fertig,
11060-            lsr            ; diff:=diff / 2
11065-            pha
11070-            tya
11075-            ror
11080-            pha
11085-            lda exp2
11090-            sta expakk
11095-            lda exp2+1
11100-            sta expakk+1
11105-            jsr setarg1
11110-            jmp nibshift
11115-arg2less    lsr
11120-            pha
11125-            tya
11130-            ror
11135-            pha
11140-            jsr setarg2
11145-nibshift    bcc bytshift
11150-            jsr shiftarg
11155-bytshift    pla
11160-            tay
11165-            pla
11170-            beq bytsh1
11175-            ldy fplen      ; high-byte der differenz<>0: dann wird
11180-            dey            ; so weit geschoben, dass auf jeden fall 0
11185-            bne fillnull   ; herauskommt
11190-bytsh1      cpy #0
11195-            beq normargsx
11200-            sty mulcount   ; differenz in byte
11205-            ldy fplen
11210-            dey            ; y = destination-pointer
11215-            tya
11220-            sec
11225-            sbc mulcount
11230-            sta mulcount   ; sourcepointer
11235-            bcc fillnull
11240-byteloop    sty ysave
11245-            ldy mulcount
11250-            lda (pt),y
11255-            ldy ysave
11260-            sta (pt),y
11265-            dey
11270-            dec mulcount
11275-            bpl byteloop
11280-fillnull    lda #0
11285-            sta (pt),y
11290-            dey
11295-            bpl fillnull
11300-normargsx rts
11305-fpmult      jsr clrfpakk
11310-            ldy #0
11315-nextdig     jsr shlong     ; fpakku in doppelter laenge
11320-            lda fparg1,y   ; um eine ziffer nach links schieben
11325-            and #$f0
11330-            beq zero1
11335-            lsr
11340-            lsr
11345-            lsr
11350-            lsr
11355-            jsr add
```

```
11360-zero1     jsr shlong     ; fpakku in doppelter laenge
11365-          lda fparg1,y   ; um eine ziffer nach links schieben
11370-          and #$f
11375-          beq zero2
11380-          jsr add
11385-zero2     iny
11390-          cpy fplen
11395-          bne nextdig
11400-          lda sign1
11405-          eor sign2
11410-          sta signakk
11415-          lda exp1
11420-          clc
11425-          adc exp2
11430-          sta expakk
11435-          lda exp1+1
11440-          adc exp2+1
11445-          sta expakk+1
11450-          bvc mulok
11455-          bmi mulerr     ; nur wenn exponent zu gross geworden, fehler
11460-          jsr zerofak
11465-          rts
11470-mulok     jsr normakku
11475-          bvs mulerr
11480-          jsr roundakk
11485-          bvs mulerr
11490-          jsr packsign
11495-mulerr    rts
11500-fpdivovl  bit overflow
11505-          rts
11510-fpdiv     lda fparg2
11515-          beq fpdivovl   ; division durch 0 fehler
11520-          jsr clrfpakk
11525-          lda sign1
11530-          eor sign2
11535-          sta signakk    ; vorzeichen des ergebnisses bestimmen
11540-          lda exp1       ; exponent des ergebnisses bestimmen
11545-          sec
11550-          sbc exp2
11555-          sta expakk
11560-          lda exp1+1
11565-          sbc exp2+1
11570-          sta expakk+1
11575-          bvc fpdiv1
11580-          bmi fpdivovl
11585-          lda #0         ; bei unterlauf kein fehler, ergebnis auf 0
11590-          sta expakk
11595-          sta expakk+1
11600-          sta signakk
11605-          clv
11610-          rts
11615-fpdiv1    ldx fplen
11620-          lda #0         ; division muss mit erhoehter mantissenlaenge
11625-          sta fparg1,x   ; ausgefuehrt werden, damit das ergebnis auch
11630-          sta fparg2,x   ; bis zur letzten stelle stimmt
11635-          inc fplen
11640-          jsr compmant   ; falls mantisse1<mantisse2,
11645-          php            ; gegeneinander verschieben und exponent
11650-          jsr arg2right  ; korrigieren
11655-          plp
11660-          bcc fpdiv2
11665-          ... incw(expakk)
11670-          jsr arg1right
11675-fpdiv2    lda dblen
11680-          sta mulcount
11685-          inc mulcount
11690-fpdiv3    jsr compmant
11695-          bcc fpdiv5
11700-          ldx fplen
11705-          sed
11710-          lda fpakku-1,x
11715-          clc
11720-          adc #$10
11725-          sta fpakku-1,x
11730-          sec
11735-fpdiv4    lda fparg1-1,x
11740-          sbc fparg2-1,x
11745-          sta fparg1-1,x
11750-          dex
11755-          bne fpdiv4
11760-          cld
11765-          jmp fpdiv3
11770-fpdiv5    dec mulcount
11775-          beq exitdiv
11780-          jsr shshort
11785-          ldy #4
11790-fpdiv6    ldx fplen
11795-          clc
11800-fpdiv7    rol fparg1-1,x
11805-          dex
11810-          bne fpdiv7
11815-          dey
11820-          bne fpdiv6
11825-          jmp fpdiv3
11830-exitdiv   dec fplen
11835-          jsr roundakk
11840-          jsr packsign
11845-          rts
11850-; zerofak: ergebnis 0 erzeugen, da unterlauf aufgetreten
11855-; overflow-flag loeschen
11860-zerofak   jsr clrfpakk
11865-          sta signakk    ; 0 noch im (prozessor-)akku
11870-          sta expakk
11875-          sta expakk+1
11880-          clv            ; kein fehler
11885-          rts
11890-; shlong und shshort schieben mantisse des akku um eine ziffer nach
11895-; links, und zwar in einfacher (shshort) oder doppelter (shlong) laenge
11900-shlong    lda dblen
11905-          bne shiftakku
11910-shshort   lda fplen
11915-shiftakku ldx #0
11920-          stx lastdig
11925-          tax
11930-          inx
11935-shiftak2  lda fpakku-1,x
11940-          asl
11945-          rol
11950-          rol
11955-          rol
11960-          pha
11965-          and #$f0       ; altes hoeherwertiges nibble isolieren
11970-          ora lastdig    ; und mit niederwertigem nibble der letzten
11975-          sta fpakku-1,x ; stelle kombinieren
11980-          pla
11985-          rol
11990-          and #$0f       ; ehemlige bits 4:7  nach 0:3 schieben
11995-          sta lastdig    ; und fuer naechste stelle merken
12000-          dex
12005-          bne shiftak2
12010-          rts
12015-roundakk  ldx fplen
12020-          lda fpakku,x
12025-          and #$f0
12030-          cmp #$50
12035-          bcc roundok
12040-          lda #0
12045-          sed
12050-round1    lda fpakku-1,x
12055-          adc #0
12060-          sta fpakku-1,x
12065-          bcc roundok
12070-          dex
12075-          bne round1
12080-          lda #$10       ; durch runden ist ueberlauf aufgetreten
12085-          sta fpakku
12090-          ... incw(expakk)
12095-roundok   cld
12100-          rts
12105-clrfpakk  ldx dblen      ; floating point akku loeschen
12110-          lda #0
12115-clrloop   sta fpakku-1,x
12120-          dex
12125-          bne clrloop
12130-          rts
12135-; addiere fpakku:=fpakku+(a)*fparg2
12140-add       sty ysave
12145-          sta mulcount
12150-          sed
12155-add1      ldy fplen
12160-          ldx dblen
12165-          clc
12170-add2      lda fparg2-1,y
12175-          adc fpakku-1,x
12180-          sta fpakku-1,x
12185-          dex
12190-          dey
12195-          bne add2
12200-          bcc addok
12205-add3      lda #0         ; uebertrag vorne aufaddieren
12210-          adc fpakku-1,x
12215-          sta fpakku-1,x
12220-          dex
12225-          bcs add3       ; bis kein uebertrag mehr vorhanden
12230-addok     dec mulcount
12235-          bne add1
12240-          cld
12245-          ldy ysave
12250-          rts
12255-; ergebnis im akku normalisieren (d.h. so verschieben, dass erste
                stelle <> 0
12260-normakku  ldx #0
12265-          ldy #0
12270-checkdig  lda fpakku,x   ; suche erstes byte in mantisse<>0
12275-          bne founddig
12280-          inx
12285-          cpx fplen
12290-          bne checkdig   ; ergebnis 0?
12295-underfl   jsr zerofak    ; bei unterlauf 0 ausgeben
12300-normok    clv
12305-          rts
12310-founddig  and #$f0       ; falls hoechstwertige ziffer im rechten
12315-          bne norm1      ; nibble, zunaechst um 1 nibble verschieben
12320-          stx xsave
12325-          jsr shlong
12330-          ldx xsave
12335-          sec            ; und exponenten korrigieren
12340-          lda expakk
12345-          sbc #1
12350-          sta expakk
12355-          lda expakk+1
12360-          sbc #0
12365-          sta expakk+1
12370-          bvs underfl
12375-norm1     txa
12380-          beq normok
12385-          asl            ; *2, da jedes byte zwei dezimalstellen gibt
12390-          sta mulcount   ; korrektur fuer exponenten
12395-          lda expakk
12400-          sec
12405-          sbc mulcount
12410-          sta expakk
12415-          lda expakk+1
12420-          sbc #0
12425-          sta expakk+1
12430-          bvs underfl
12435-          ldy #0         ; akku byteweise nach links verschieben
12440-norm2     lda fpakku,x
12445-          sta fpakku,y
12450-          iny
12455-          inx
12460-          cpx fplen
12465-          bcc norm2      ; und rechts mit nullen auffuellen
12470-          beq norm2
12475-          lda #0
12480-norm3     sta fpakku,y
12485-          iny
12490-          cpy fplen
12495-          bcc norm3
12500-          bcs normok
12505-; fparg1 um eine ziffer (1 nibble) nach rechts verschieben
12510-arg1right jsr setarg1
```

Listing 10. FPBCD.HY (Fortsetzung)

```
12515-          jmp shiftarg
12520-; fparg2 um eine ziffer (1 nibble) nach rechts verschieben
12525-arg2right jsr setarg2
12530-; argument um ein nibble nach rechts schieben
12535-; pt/pt+1 zeigt auf mantisse des arguments
12540-shiftarg  sty ysave
12545-sha1      ldy #0
12550-          sty lastdig
12555-sha2      lda (pt),y
12560-          lsr            ; byte zyklisch vertauschen:
12565-          ror            ; bit 4:7 -> bit 0:3
12570-          ror            ; bit 0:3 -> bit 5:7 und carry
12575-          ror
12580-          pha
12585-          and #$0f       ; altes hoeherwertiges nibble isolieren
12590-          ora lastdig    ; und mit niederwertigem nibble der letzten
12595-          sta (pt),y     ; stelle kombinieren
12600-          pla
12605-          ror
12610-          and #$f0       ; ehemlige bits 0:3  nach 4:7 schieben
12615-          sta lastdig    ; und fuer naechste stelle merken
12620-          iny
12625-          cpy fplen
12630-          bcc sha2
12635-          ldy ysave
12640-          rts
12645-; zeiger auf fparg1 oder fparg2 setzen
12650-setarg1   ldx #<(fparg1)
12655-          stx pt
12660-          ldx #>(fparg1)
12665-          stx pt+1
12670-          rts
12675-setarg2   ldx #<(fparg2)
12680-          stx pt
12685-          ldx #>(fparg2)
12690-          stx pt+1
12695-          rts
12700-; vergleicht abs(fparg1) und abs(fparg2)
12705-; ergebnis: carry := abs(fparg1)>=abs(fparg2)
12710-compabs   lda exp1+1
12715-          cmp exp2+1
12720-          bne diffexp
12725-          lda exp1
12730-          cmp exp2
12735-          beq compmant
12740-          sec
12745-          bpl compexit
12750-          clc
12755-diffexp   rts
12760-; vergleicht mantisse von fparg1 mit mantisse von fparg2,
          ergebnis s. compabs
12765-compmant  ldx #0
12770-comploop  lda fparg1,x
12775-          cmp fparg2,x
12780-          bne compexit
12785-          inx
12790-          cpx fplen
12795-          bne comploop
12800-compexit  rts            ; eq bei gleichheit,
          carry <=> abs(fparg1)>=abs(fparg2)
12805-overflow  .by $ff
12810-;         genauigkeit
12815-fplen     .by 0
12820-dblen     .by 0
12825-;         hilfsspeicher
12830-lastdig   .by 0
12835-xsave     .by 0
12840-ysave     .by 0
12845-mulcount  .by 0
12850-currentlen .by 0
12855-; 1. argument
12860-sign1     .by 0
12865-exp1      .wo 0
12870-fparg1    ... ds(maxlen)
12875-; 2. argument
12880-sign2     .by 0
12885-exp2      .wo 0
12890-fparg2    ... ds(maxlen)
12895-; ergebnis
12900-signakk   .by 0
12905-expakk    .wo 0
12910-fpakku    ... ds(maxlen*2)
```

Listing 10. FPBCD.HY (Schluß)

```
100  -.ba $c000
105  -.gl stack=$cfff
110  -          jmp main
```

Listing 11. UPN1.HY

```
15000-;
15005-; upn-rechner, auf fpbcd-routinen basiert
15010-; arbeitet mit stack vom programmende bis $cfff
15015-; befehle:
15020-; links pfeil  obersten beiden elemente vertauschen
15025-; ?            oberstes element ausgeben
15030-; d            oberstes element entfernen
15035-; ^            oberstes element verdoppeln
15040-; x            beenden
15045-; +,-,*,/      rechnen
15050-; alles andere: zahl einlesen
15055-; zahlen mit vorzeichen muessen mit einer leerstelle eingeleitet werden,
15060-; damit das vorzeichen nicht als subtraktionsbefehl missdeutet wird
15065-;
15070-main      lda #14        ; mantissenlaenge 28 stellen
15075-          jsr setprecision
15080-          jsr initstk
15085-mainloop  jsr prompt
15090-          jsr basin
15095-          cmp #cr
15100-          beq mainloop
15105-          cmp #"?"
15110-          bne noprint
15115-          jmp printcom
15120-noprint   cmp #"x"
15125-          bne nobreak
15130-          rts
15135-nobreak   cmp #"^"
15140-          bne noenter
15145-          lda nstack
15150-          bne enterok
15155-          jmp emptystk
15160-enterok   jsr popstk
15165-          jsr pushstk
15170-          jsr pushstk
15175-          jmp mainloop
15180-noenter   cmp #"d"
15185-          bne nodrop
15190-          lda nstack
15195-          beq emptystk
15200-          jsr popstk
15205-          jmp mainloop
15210-nodrop    cmp #$5f       ; pfeil nach links
15215-          bne noswap
15220-          ldx nstack
15225-          cpx #2
15230-          bcc emptystk
15235-          jsr popstk
15240-          jsr swapnumber
15245-          jsr popstk
15250-          jsr swapnumber
15255-          jsr pushstk
15260-          jsr swapnumber
15265-          jmp printpush
15270-noswap    cmp #"+"
15275-          beq foundop
15280-          cmp #"-"
15285-          beq foundop
15290-          cmp #"*"
15295-          beq foundop
15300-          cmp #"/"
15305-          beq foundop
15310-          tax
15315-          ... ldayi(number)
15320-          jsr inputfp
15325-          jsr pushstk
15330-          bcc mainloop
15335-          ... ldayi(sover)
15340-error     jsr prints
15345-          jmp main
15350-sover     .tx "*** stack overflow ***"
15355-.by 0
15360-printcom  lda nstack
15365-          bne printstack
15370-emptystk  ... ldayi(sempty)
15375-          jmp error
15380-sempty    .tx "*** stack empty ***"
15385-.by 0
15390-printstack jsr popstk
15395-printpush jsr pushstk    ; zahl wieder zurueck auf stack
15400-          ... ldayi(number)
15405-          jsr printfp
15410-          jmp mainloop
15415-foundop   ldx nstack
15420-          cpx #2
15425-          bcc emptystk
15430-          pha
15435-          jsr popstk
15440-          ... ldayi(number)
15445-          jsr loadarg2
15450-          jsr popstk
15455-          ... ldayi(number)
15460-          jsr loadarg1
15465-          pla
15470-          cmp #"+"
15475-          bne noplus
15480-          jsr fpadd
15485-          jmp pushres
15490-noplus    cmp #"-"
15495-          bne nominus
15500-          jsr fpsub
15505-          jmp pushres
15510-nominus   cmp #"*"
15515-          bne notimes
15520-          jsr fpmult
15525-          jmp pushres
15530-notimes   jsr fpdiv
15535-pushres   bvc resok
15540-          ... ldayi(aover)
15545-          jmp error
15550-aover     .tx "*** arithmetic overflow ***"
15555-.by 0
15560-resok     ... ldayi(number)
15565-          jsr storeakk
15570-          jmp printpush
15575-prompt    jsr crlf
15580-          ldy nstack
15585-          iny
15590-          sed
15595-          lda #$99
15600-todec     clc
15605-          adc #$01
15610-          dey
15615-          bne todec
15620-          cld
15625-          pha
15630-          lsr
15635-          lsr
15640-          lsr
```

```
15645-          lsr
15650-          jsr printdd
15655-          pla
15660-          jsr printdd
15665-          lda #">"
15670-          jsr basout
15675-          rts
15680-printdd   and #$0f
15685-          ora #"0"
15690-          jmp basout
15695-initstk   lda #0
15700-          sta nstack
15705-          ... movewi(stack,sp)
15710-          rts
15715-popstk    lda nstack
15720-          bne pop1
15725-          sec
15730-          rts
15735-pop1      dec nstack
15740-          ldx fplen
15745-pop2      jsr dopull
15750-          sta number+1,x ; mantisse
15755-          dex
15760-          bne pop2
15765-          jsr dopull
15770-          sta number+1   ; exponent high-byte und vorzeichen
15775-          jsr dopull
15780-          sta number     ; exponent low-byte
15785-          clc
15790-          rts
15795-pushstk   lda nstack
15800-          cmp #<(maxstack)
15805-          bcc push1
15810-          rts
15815-push1     inc nstack
15820-          lda number
15825-          jsr dopush
15830-          lda number+1
15835-          jsr dopush
15840-          ldx #0
15845-push2     lda number+2,x
15850-          jsr dopush
15855-          inx
15860-          cpx fplen
15865-          bcc push2
15870-          clc
15875-          rts
15880-swapnumber ldx fplen     ; benutzt fpakku als zwischenspeicher
15885-          inx
15890-swaploop  lda number,x
15895-          pha
15900-          lda fpakku,x
15905-          sta number,x
15910-          pla
15915-          sta fpakku,x
15920-          dex
15925-          cpx #$ff
15930-          bne swaploop
15935-          rts
15940-dopush    ... pushb(sp)
15945-          rts
15950-dopull    ... pullb(sp)
15955-          rts
15960-inputfp   ... stay(pt)   ; ziel-zeiger fuer fertige bcd-zahl merken
15965-          txa
15970-          pha
15975-          jsr clearbuf   ; mantisse, exponent und vorzeichen loeschen
15980-          pla
15985-          jsr skipspace  ; leerzeichen ueberlesen
15990-          jsr getmsign   ; vorzeichen der mantisse einlesen
15995-          jsr skipzero   ; vornullenunterdrueckung fuer mantisse
16000-          jsr getipart   ; vorkommaanteil holen
16005-          cmp #"."
16010-          bne tryexp
16015-          jsr basin
16020-          jsr getfpart   ; nachkommaanteil holen
16025-tryexp    cmp #"e"
16030-          bne convertf   ; kein exponent: zahl ist fertig eingelesen
16035-          jsr basin
16040-          jsr getesign   ; vorzeichen des exponenten holen
16045-          jsr skipzero   ; vornullenunterdrueckung des exponenten
16050-          jsr getexp     ; exponent holen und mit der
16055-          jsr makeexp    ; anzahl der vor- und nachkommastellen kombinieren
16060-convertf  jsr copysex    ; sign und exponent kopieren
16065-          jsr copymant   ; mantisse kopieren und packen
16070-          rts
16075-; ergibt carry cler, wenn ziffer im akku, carry set wenn keine ziffer
16080-digit     cmp #"0"
16085-          bcc nodigit
16090-          cmp #$3a       ;"9"+1
16095-          rts
16100-nodigit   sec
16105-          rts
16110-; mantissen-vorzeichen lesen und merken, wenn vorhanden
16115-getmsign  jsr getsign
16120-          bne nomsign
16125-          stx sign
16130-nomsign   rts
16135-; exponenten-vorzeichen lesen und merken, wenn vorhanden
16140-getesign  jsr getsign
16145-          bne noesign
16150-          stx esign
16155-noesign   rts
16160-; teste, ob akku vorzeichen enthaelt, wenn ja, zero-flag setzen,
16165-; im x-register ergebnis (0 oder $ff) uebergeben und naechstes zeichen lesen
16170-getsign   ldx #$ff
16175-          cmp #"-"
16180-          beq gotsign
16185-          inx
16190-          cmp #"+"
16195-gotsign   bne nosign
16200-          php            ; zero-flag merken
16205-          jsr basin
16210-          plp
16215-nosign    rts
16220-; fuehrende nullen unterdruecken
16225-skipzero  cmp #"0"
16230-          bne nozero
16235-          jsr basin
16240-          jmp skipzero
16245-nozero    rts
16250-clearbuf  ldx dblen
16255-          lda #"0"
16260-initmant  sta mantisse-1,x
16265-          dex
16270-          bne initmant
16275-          stx exp
16280-          stx exp+1
16285-          stx eexp       ; hilfs-exponent
16290-          stx eexp+1
16295-          stx esign      ; vorzeichen des exponents
16300-          stx sign
16305-          rts
16310-; leerzeichen, geshiftete leerzeichen und return ueberlesen
16315-skipspace cmp #" "
16320-          beq isspace
16325-          cmp #$a0       ;$80+" "
16330-          beq isspace
16335-          cmp #$0d
16340-          beq isspace
16345-          rts
16350-isspace   jsr basin
16355-          jmp skipspace
16360-; vorkommastellen holen, dabei exponent mitzaehle
16365-getipart  ldx #0
16370-ipart     jsr digit
16375-          bcs ipartex
16380-          cpx dblen      ; passt zeichen noch in buffer?
16385-          bcs noistore
16390-          sta mantisse,x
16395-          inx
16400-noistore  inc exp        ; vorkommastellen mitzaehlen
16405-          bne nextidig
16410-          inc exp+1
16415-nextidig  jsr basin
16420-          jmp ipart
16425-ipartex   rts
16430-; nachkomma-anteil holen, dabei beachten, ob ueberhaupt schon
16435-; ziffern in der mantisse vorliegen, wenn nicht, erst mal
16440-; wieder fuehrende nullen unterdruecken
16445-getfpart  cpx #0         ; wenn noch gar keine ziffern gefunden,
16450-          bne contmant   ; erstmal wieder vornullen unterdrucken
16455-skipfz    cmp #"0"
16460-          bne contmant   ; um z.b. die zahl 0.005 zu 0.5 * 10 ^ -2
16465-          lda exp
16470-          bne decexp1
16475-          dec exp+1
16480-decexp1   dec exp        ; zu normalisieren
16485-          jsr basin
16490-          jmp skipfz
16495-contmant  jsr digit
16500-          bcs fpartex    ; keine ziffer mehr -> mantisse fertig
16505-          cpx dblen      ; noch platz?
16510-          bcs nofstore
16515-          sta mantisse,x
16520-          inx
16525-nofstore  jsr basin
16530-          jmp contmant
16535-fpartex   rts
16540-; exponent einlesen, keine fehlerabfrage, exponent muss
16545-; im bereich von 0 bis 16383 liegen!
16550-getexp    jsr digit
16555-          bcs gotexp
16560-          and #$0f
16565-          ldx #10        ; eexp:=eexp*10+ziffer primitiv aber kurz
16570-          ldy #0
16575-expmal10  clc
16580-          adc eexp
16585-          pha
16590-          tya
16595-          adc eexp+1
16600-          tay
16605-          pla
16610-          dex
16615-          bne expmal10
16620-          sta eexp
16625-          sty eexp+1
16630-          jsr basin
16635-          jmp getexp
16640-gotexp    rts
16645-; explizit angegebenen expont (eexp) mit position des dezimalpunktes
16650-; zu einer normalisierten darstellung kombinieren
16655-makeexp   lda exp
16660-          bit esign
16665-          bpl addexp
16670-          sec
16675-          sbc eexp
16680-          sta exp
16685-          lda exp+1
16690-          sbc eexp+1
16695-          sta exp+1
16700-          rts
16705-addexp    clc
16710-          adc eexp
16715-          sta exp
16720-          lda exp+1
16725-          adc eexp+1
16730-          sta exp+1
16735-          rts
16740-; vorzeichen und exponent an den gewuenschten platz kopieren
16745-; und dabei packen (vorzeichen der mantisse -> bit15 des exponenten
16750-copysex   ldy #0
16755-          lda exp
16760-          sta (pt),y
16765-          iny
16770-          lda exp+1
16775-          and #$7f
16780-          bit sign
16785-          bpl copysex1
16790-          ora #$80
16795-copysex1  sta (pt),y
```

Listing 12. UPN2.HY

```
16800-          iny
16805-          rts
16810-; mantisse packen (immer 2 ziffern in ein byte) und kopieren
16815-copymant  ldx #0
16820-cmantlo   lda mantisse,x
16825-          and #$0f
16830-          asl
16835-          asl
16840-          asl
16845-          asl
16850-          sta sign        ; zwischenspeicher
16855-          lda mantisse+1,x
16860-          and #$0f
16865-          ora sign
16870-          sta (pt),y
16875-          iny
16880-          inx
16885-          inx
16890-          cpx dblen
16895-          bcc cmantlo
16900-          rts
16905-printfp   ... stay(pt)
16910-          ldy #0
16915-          jsr prmsign
16920-          jsr pripart
16925-          jsr prmant
16930-          jsr presign
16935-          jsr prexp
16940-          rts
16945-prmsign   iny             ; mantissen-vorzeichen ist in bit 7 des
16950-          lda (pt),y      ; exponent untergebracht
16955-          dey
16960-          rol             ; vorzeichen in das carry-flag retten
16965-          lda #" "
16970-          bcc doprsign
16975-          lda #"-"
16980-doprsign  jsr basout
16985-          iny
16990-          iny             ; y auf mantisse richten
16995-          rts
17000-pripart   lda #"0"        ; normalisierte darstellung faengt immer
17005-          jsr basout      ; mit "0." an
17010-          lda #"."
17015-          jsr basout
17020-          rts
17025-prmant    ldx fplen
17030-prpair    lda (pt),y      ; 2 ziffern entpacken und ausgeben
17035-          pha
17040-          and #$f0
17045-          lsr
17050-          lsr
17055-          lsr
17060-          lsr
17065-          ora #"0"
17070-          jsr basout
17075-          pla
17080-          and #$0f
17085-          ora #"0"
17090-          jsr basout
17095-          iny
17100-          dex
17105-          bne prpair
17110-          rts
17115-; vorzeichen des exponenten bestimmen und ausdrucken, exponent
17120-; positiv machen, wird im x (low) und a (high) hinterlassen
17125-presign   lda #"e"
17130-          jsr basout
17135-          ldy #0
17140-          lda (pt),y
17145-          tax             ; exponent-low
17150-          iny
17155-          lda (pt),y      ; exponent-high
17160-          and #$7f        ; isolieren
17165-          cmp #$40
17170-          bcc exppos
17175-          eor #$7f
17180-          pha
17185-          txa
17190-          eor #$ff
17195-          adc #0
17200-          tax
17205-          php
17210-          lda #"-"
17215-          jsr basout
17220-          plp
17225-          pla
17230-          adc #0
17235-          rts
17240-exppos    pha
17245-          lda #"+"
17250-          jsr basout
17255-          pla
17260-          rts
17265-; zwei-byte-exponent ausgeben
17270-prexp     stx pt
17275-          sta pt+1        ; als zwischenspeicher
17280-          lda #0
17285-          sta expflag     ; noch keine ziffer ausgegeben
17290-          ldy #4
17295-prexp0    lda #"0"        ; naechste ziffer
17300-          sta dig
17305-prexp1    lda pt
17310-          sec
17315-          sbc lowtab,y
17320-          tax
17325-          lda pt+1
17330-          sbc hightab,y
17335-          bcc prexp2
17340-          inc dig
17345-          dec expflag     ; merke, dass ziffer<>0 gefunden
17350-          stx pt
17355-          sta pt+1
17360-          bcs prexp1
17365-prexp2    bit expflag
17370-          bpl prexp3
17375-          lda dig
17380-          jsr basout
17385-prexp3    dey
17390-          bpl prexp0      ; naechste ziffer
17395-          bit expflag
17400-          bmi prexp4      ; wenn noch gar nichts ausgegeben,
17405-          lda #"0"        ; jetzt eine 0 ausgeben
17410-          jsr basout
17415-prexp4    rts
17420-lowtab    .by <(1),<(10),<(100),<(1000),<(10000)
17425-hightab   .by >(1),>(10),>(100),>(1000),>(10000)
17430-; variable fuer i/o-routinen
17435-sign      .by 0
17440-esign     .by 0
17445-exp       .wo 0
17450-eexp      .wo 0
17455-mantisse  ... ds(maxlen*2)
17460-dig       .by 0
17465-expflag   .by 0
17470-number    ... ds(maxlen*2)
17475-nstack    .by 0
17480-.eq maxstack=(stack-nstack)/(maxlen+2)
17485-.en
```

Listing 12. UPN2.HY (Schluß)

```
10000-start     ... mopen(1,dev2,name,len); file oeffnen
10005-          ... dstat(dev,buffer); floppy-status pruefen
10010-          beq fileok
10015-          ldx #0          ; wenn fehler, meldung ausgeben
10020-errmsg    lda buffer,x
10025-          beq exit        ; meldung zuende: file schliessen
10030-          jsr basout
10035-          inx
10040-          bne errmsg      ; unbedingter sprung
10045-fileok    ldx #1
10050-          jsr chkin       ; eingabe von file vorbereiten
10055-list      jsr basin
10060-          jsr basout      ; file auf bildschirm listen
10065-          jsr stopeq      ; bei stoptaste abbrechen
10070-          beq exit
10075-          jsr readst      ; solange file nicht zuende,
10080-          beq list        ; weiterlesen
10085-exit      jsr clrchn      ; standard ein/ausgabe
10090-          ... mclose(1)   ; file schliessen
10095-          rts
10100-name      .tx "myfile"    ; konstanter filename
10105-len       .by (len-name)  ; laenge des filenamens ausrechnen
10110-dev       .by 8           ; geraetenummer der 1541
10115-buffer    .wo 0,0,0,0,0,0,0,0; ab hier 33 byte platz lassen
10120-.wo 0,0,0,0,0,0,0,0
10125-.wo 0,0,0,0,0,0,0,0
10130-.wo 0,0,0,0,0,0,0,0
10135-.by 0
10140-.en
```

Listing 13. EX1.HY

```
10000-start     jsr writes
10005-.tx "dies ist ein beispiel"
10010-.by cr
10015-.tx "fuer die string-ein/ausgabe"
10020-.by cr
10025-.tx "bitte string eingeben:"
10030-.by 0
10035-          ... minputs(buffer,80)
10040-          cpy #0
10045-          beq nichts
10050-          jsr writes
10055-.by cr
10060-.tx "sie haben '"
10065-.by 0
10070-          ... mprints(buffer)
10075-          jsr writes
10080-.tx "'eingegeben"
10085-.by cr,0
10090-          rts
10095-nichts    jsr writes
10100-.by cr
10105-.tx "das war gar nichts!"
10110-.by cr
10115-.tx "bitte noch mal:"
10120-.by 0
10125-          jmp again
10130-buffer    .by 0
10135-.en
```

Listing 14. EX2.HY

```
10000-.gl stack=$24
10005-initstack ... movewi($9fff,stack); stack wächst nach unten
10010-          rts             ; ab $9fff
10015-dopush    ... pushw(stack)
10020-          rts
10025-dopull    ... pullw(stack)
10030-          rts
10035-check     ... lday(stack); maximal 4 k stack
10040-          ... cmpwi($9000)
10045-          bcc overflow
10050-          rts
10055-overflow  jsr prints
10060-.tx "stack overflow"
10065-.by 0
10070-          brk
10075-.en
```

Listing 15. EX3.HY

Von Basic zu Assembler

Vom vertrauten Basic zum Programmieren in Assembler ist es gar nicht so weit. Wir werden langsam anfangen und uns von grundlegenden Schritten bis zu komplexen Problemlösungen vorarbeiten.

Man kann wohl getrost davon ausgehen, daß ein großer Teil der Commodore-64-Benutzer Basic beherrscht. Vermutlich hat es Sie aber auch schon öfters gereizt, die langsamen Programme durch Assembler-Routinen zu beschleunigen. Aber bis zum eigenen Maschinenprogramm ist es häufig noch ein holpriger Weg. Ein Führer, der Ihnen den Pfad aus den weiten Basic-Ebenen auf die Gipfel der Assembler-Programmierung zeigt, ist dieser Kurs.

Zunächst halten wir uns an Bekanntes: Basic-Befehle sollen auf ihre Entsprechungen in Assembler untersucht werden. Oft aber streifen wir die Fesseln der Hochsprache ab und lernen allerlei Assembler-Techniken kennen. Wo es uns angemessen erscheint, nehmen wir uns die Freiheit, die computerinterne Firmware zu benutzen. Alles in allem sollen Sie durch Training zum vertrautem Umgang mit Assembler gelangen.

Basic contra Assembler

Basic macht uns den Umgang mit dem Computer relativ leicht: Wir brauchen uns kaum um die Verwaltung von Variablen zu kümmern. Benötigen wir zum Beispiel eine neue Stringvariable, dann richtet Basic sie beim ersten Auftauchen automatisch ein. Sehr entgegenkommend ist Basic auch, was den Verkehr mit Peripherie jeder Art angeht: Ein Zeichen von der Tastatur anzunehmen, ist mit GET oder INPUT recht einfach. Irgend etwas auf dem Drucker zu schreiben oder auf die Diskette: mit OPEN und dem PRINT# geht's reibungslos und ohne Gedanken an die Busverwaltung. Die Hochsprache »Basic« nimmt uns vieles ab, worum wir uns in Assembler kümmern müssen.

Andererseits befindet man sich etwa in der Situation eines Bauherrn, der sich ein Haus mit Fertigteilen aufstellen lassen möchte: Jeder Wunsch, der vom Standard abweicht, ist nicht möglich oder wird sehr teuer. Basic-Erweiterungen sind dann mit einer großen Sammlung von Fertigteil-Formen zu vergleichen: Man muß alle kaufen, obwohl man nur einige für den Sonderwunsch braucht.

Der Vergleich fängt nun etwas zu hinken an: Die Fertigteile setzen sich nämlich bei genauerem Hinsehen aus kleinen Einzelteilen (Ziegelsteinen) zusammen. Wenn man also auf die vorgefertigten Großteile weitgehend verzichtet, dafür aber den Umgang mit Ziegelsteinen (Assemblerbefehlen) beherrscht, kann man sich genau das individuelle Haus bauen, wie bizarr es auch immer aussehen mag.

Wenn man sich das Verarbeiten eines Basic-Programmes etwas genauer ansieht, dann versteht man die Schwerfälligkeit, mit der vieles geschieht. Vom Augenblick des Einschaltens an läuft ein Maschinenprogramm im Computer: Der Interpreter. Jeder Tastendruck, jeder Basic-Befehl, der diesem in die Hände fällt, wird untersucht und führt zur Abarbeitung eines auf diesen Befehl zugeschnittenen Assemblerprogrammes.

Ein Basic-Befehl wie beispielsweise PRINT kann ganz verschiedene Reaktionen erfordern: Wenn vorher ein CMD-Kommando erfolgt war, findet die Ausgabe nicht auf dem Bildschirm statt. Wenn das nicht der Fall war, kann der Bildschirm an einer anderen Stelle als im Normalfall liegen.

Der PRINT-Befehl im Vergleich

Das, was auszudrucken ist, kann ein String sein, eine Integerzahl oder eine Fließkommavariable, allerlei Fehlerquellen sind abzufangen etc. Das zu PRINT gehörende Assembler-Programm muß all diese Möglichkeiten berücksichtigen, wie selten sie auch angesprochen werden.

Bei einem individuellen Assembler-Programm wissen wir dagegen, was wir wie ausgeben wollen. Unser eigener »PRINT«-Befehl wird nur das enthalten, was unbedingt notwendig ist, der ganze unnötige Ballast wird von uns nicht programmiert. Allerdings sind wir dann auch voll verantwortlich für die einwandfreie Funktion. Wir müssen beispielsweise dafür sorgen, daß bei Zugriffen auf Register

oder Speicher dort der richtige Wert zum richtigen Zeitpunkt im erlaubten Format zur Verfügung steht. Das ist wie im täglichen Leben: Die größere Freiheit legt uns mehr Verantwortung auf und schafft uns andererseits ungeahnte Möglichkeiten, Ideen zu verwirklichen.

Einige Vorbemerkungen

Für die ersten Beispielprogramme werde ich den Assembler »Hypra-Ass« verwenden. Er wurde im 64'er-Sonderheft 8/85 abgedruckt und ist auch auf der entsprechenden Programmservice-Diskette erhältlich. Besonders interessant wird für Sie in diesem Zusammenhang auch der »Giga-Ass« aus dieser Ausgabe.

Weil man aber auch einen Monitor braucht, findet zu diesem Zweck weiterhin der SMON hier seinen Platz. Ich verwende eine Version, die bei $9000 beginnt, um ab $C000 Raum zu lassen für den Reassembler zu Hypra-Ass (Sonderheft 8/85). Das ist ein nützliches Programm, mit dem man Maschinencode aus dem Speicher wieder in einen Quelltext umwandeln kann. Mittels des Hypra-Ass ist dieser Quelltext dann bearbeitbar. Mit diesem kompletten Instrumentarium sind wir allen Aufgaben gewachsen. Quellcode von Hypra-Ass können Sie übrigens in das Giga-Ass-Format konvertieren. Beim Arbeiten mit Giga-Ass sollte der SMON bei $C000 beginnen, da Giga-Ass im Speicherbereich $8000 bis $9FFF liegt. Im weiteren wollen wir uns aber auf den Hypra-Ass beziehen.

Einfache Schleifen

Eine der meistgebrauchten Strukturen in Basic und auch eine der wichtigsten in Assembler ist die Programmschleife. Als »einfache« Schleife bezeichne ich solche, die zum Zählen nur 1 Byte erfordern, also maximal 256 Durchläufe erlauben. Die verschiedenen Möglichkeiten sehen wir uns anhand von Verzögerungsschleifen an, die zunächst einmal nichts anderes tun, als zu zählen und Zeit zu verbrauchen (welch ein Luxus!). Die einfachste Variante lautet in Basic etwa:

```
10 FOR I = 0 TO 255
20 NEXT I
```

In Listing 1 finden Sie die »Übersetzung« in Assembler. Sie können sowohl das Y-Register (Variante 1) als auch das X-Register (Variante 2) zum Zählen verwenden. In Zeile 5 finden Sie .LI 1,4. Das ist ein Pseudobefehl (also kein 6502-Befehl), der die Ausgabe des Protokolls über den Drucker bewirkt. Zeile 40 enthält durch .BA $5000 wieder einen Pseudobefehl. Damit legt man fest, von welcher Adresse an der Maschinencode in den Speicher gelegt werden soll. Die Zeilen 50 bis 90 sind unser Assemblerprogramm. Zuerst wird ein Startwert 0 in das Y-Register geschrieben und dieses dann in der Zeile mit dem Label um 1 hochgezählt.

Falls Ihnen der Ausdruck Label nicht geläufig ist: Natürlich kann man statt dessen die Adresse 5002 hinter den BCC-Befehl in Zeile 80 schreiben – so haben wir das ja bisher immer mit dem SMON-Assembler getan. Das hätte aber im Quelltext, den wir hier schreiben, den Nachteil, daß wir diese Adresse jedesmal ändern müßten, wenn wir uns entschlößen, mit dem Pseudobefehl .BA den Programmstart zu verlegen. Indem wir aber diese Zeile durch das Label-Kennzeichen markieren, merkt sich der Hypra-Ass die dazugehörige Zeilennummer und rechnet sie beim Assemblieren automatisch in die richtige Sprungadresse um.

```
HYPRA-ASS  ASSEMBLERLISTING:

            5   -          .LI 1,4
;*** VERZOEGERUNGSSCHLEIFE VARIANTEN 1 UND 2 ***
; X- ODER Y-REGISTER ALS ZAEHLER
;
            40  -          .BA $5000
5000 A000   :50 -          LDY #$00      ;BZW. LDX
5002 C8     :60 -LABEL     INY           ;     INX
5003 C0FF   :70 -          CPY #$FF      ;     CPX
5005 90FB   :80 -          BCC LABEL     ;WENN <255
5007 00     :90 -          BRK
            100 -          .SY 1,4

SYMBOLS IN ALPHABETICAL ORDER:

LABEL         = $5002

END OF ASSEMBLY  0:14.6
BASE = $5000  LAST BYTE AT $5007
```

Listing 1. Etwas verzögern mit Variante 1 und 2

```
10   -          .LI 1,4
20   -          .BA $5000
30   -;*** VERZOEGERUNGSSCHLEIFE VARIANTE 4 ***
40   -;VARIABLER ENDWERT
50   -;Y-REGISTER ALS ZAEHLER
60   -;
70   -          LDA #$20       ;DAS IST DEZIMAL 32
80   -          STA $FA        ;ENDWERT SPEICHERN
90   -;
100  -          LDY #$00       ;Y-REGISTER INITIALISIEREN
110  -LABEL     INY
120  -          CPY $FA        ;ENDWERT ERREICHT?
130  -          BCC LABEL      ;NEIN:DANN WEITERZAEHLEN
140  -          BRK
150  -;
160  -          .SY 1,4
170  -          .ST
```

Listing 2. Verschieden verzögern mit Variante 4

```
10   -          .LI 1,4
20   -          .BA $5000
30   -;*** VERZOEGERUNGSSCHLEIFE VARIANTE 5 ***
40   -;VARIABLER ENDWERT
50   -;SPEICHERSTELLE ALS ZAEHLER
60   -;
70   -          LDA #$20       ;DAS IST DEZIMAL 32
80   -          STA $FA        ;ENDWERT SPEICHERN
90   -;
100  -          LDA #$00       ;ZAEHLER INITIALISIEREN
110  -          STA $5100      ;DA IST ER:UNSER ZAEHLER
120  -;
130  -LABEL     INC $5100
140  -          LDA $5100
150  -          CMP $FA        ;VERGLEICHEN MIT ENDWERT
160  -          BCC LABEL      ;WEITERZAEHLEN WENN <> ENDWERT
170  -          BRK
180  -          .SY 1,4
190  -          .ST
```

Listing 3. Dies ist die Variante 5

```
10   -          .LI 1,4
20   -          .BA $5000
30   -;*** VERZOEGERUNGSSCHLEIFE VARIANTE 6 ***
40   -;ABWAERTS ZAEHLEN (Y-REGISTER)
50   -;
60   -          LDY #$FF       ;STARTWERT NACH Y
70   -LABEL     DEY
80   -          BNE LABEL      ;WEITER BIS Y = 0
90   -          BRK
100  -;
110  -          .SY 1,4
120  -          .ST
```

Listing 4. Rückwärts verzögern mit Variante 6

```
10   -          .LI 1,4
20   -          .BA $5000
30   -;*** VERZOEGERUNGSSCHLEIFE VERSION 7 ***
40   -;ABWAERTS ZAEHLEN ($FA ALS ZAEHLER)
50   -;
60   -          LDA #$20       ;STARTWERT IN ZAEHLER
70   -          STA $FA        ;SCHREIBEN
80   -LABEL     DEC $FA
90   -          BNE LABEL      ;WEITER BIS $FA = 0
100  -          BRK
110  -;
120  -          .SY 1,4
130  -          .ST
```

Listing 5. Hier haben wir die Variante 7

Ein weiterer Vorteil ist, daß man zu Dokumentationszwecken jede wichtige Adresse auf diese Weise markieren und sich am Schluß durch eine Symboltabelle ausgeben lassen kann. Besonders bei langen Programmen, in denen man dann sinnvolle Labelnamen verwendet (beispielsweise DRUCKEN am Anfang des Programmteils, das einen Ausdruck steuert), kann das eine unschätzbare Hilfe sein.

In unserem Programm in Listing 1 geht es weiter mit dem Vergleich, ob im Y-Register nach der Erhöhung schon $FF erreicht wurde. Ist das nicht der Fall, dann ist das Carry-Bit frei und der Programmablauf verzweigt zurück zur Labelzeile. Ansonsten ist die Verzögerungsschleife beendet und mit dem BRK meldet sich der SMON, den Sie zu diesem Zeitpunkt natürlich im Speicher haben sollten (vergessen Sie nicht, den SMON zumindest einmal zu starten mit SYS »startadresse«, damit bei einem BRK in den SMON gesprungen wird).

Falls Sie das Programm durch SYS $5000 vom Hypra-Ass aus gestartet haben (und nicht durch G 5000 aus dem SMON), finden Sie sich ebenfalls im Monitor wieder. Fast alle Beispielprogramme in diesem Kurs werden mit BRK enden. Der Grund dafür ist, daß es oft interessant ist, die Register nach dem Programmende zu beobachten. Sollten Sie ohne Monitor arbeiten wollen, dann müßten Sie statt dessen ein RTS einsetzen. Hinter dem eigentlichen Programm finden Sie .SY 1,4. Auch das ist ein Pseudobefehl, der die Ausgabe der Symboltabelle über den Drucker bewirkt. Nicht sichtbar ist ein Befehl .ST, mit dem die Assemblierung beendet wird. Einige interessante Angaben besorgt uns der Hypra-Ass noch nach der kurzen Symboltabelle: Eine Zeitangabe und den Bereich, in dem der Maschinencode nun nach der Assemblierung zu finden ist. Falls Sie diesen Objektcode (so nennt man den Maschinencode auch häufig) speichern wollen (vom Monitor aus mit dem S-Kommando möglich), dann brauchen Sie diese Angaben. Unser Programm würde dann so abgespeichert:

S"OBJ.VERZ.VAR1" 5000 5008

(Man muß immer ein Byte zur Endadresse hinzurechnen beim Speichern des Objektcode). Eine andere Möglichkeit, den Objektcode auf Diskette zu speichern: Nach dem .LI 1,4 in der nächsten Zeile folgenden Befehl einsetzen:

10 -.OB"OBJ.VERZ.VAR1,p,w

Jetzt wird nach dem Starten des Assemblierens mit RUN automatisch das Maschinenprogramm gespeichert.

Die weiteren Programmbeispiele werde ich nicht so erschöpfend erklären. Nur wenn neue Pseudobefehle verwendet werden oder eine neue Programmstruktur es erfordert, geht's nochmal in die Tiefe. Um etwas Platz zu sparen, wurden die folgenden Programme nicht mit dem .LI - Befehl aus dem Drucker ausgegeben, sondern mit

OPEN 1,4:CMD1
/E

Dadurch werden die Adressen mit den Hex-Codes der Maschinenbefehle nicht gedruckt, sondern nur das Listing, wie es auch auf dem Bildschirm zu sehen ist.

Häufig tritt in Schleifen der Fall ein, daß weder das Y- noch das X-Register zur Verfügung stehen. Sie dienen dann anderweitig schon als Index. Statt dessen kann ebensogut eine Speicherstelle den Zähler bilden, wie in dieser Variante 3:

```
      LDA   #$00
      STA   $FB          ;$FB ist Zähler
LABEL INC   $FB
      LDA   $FB
      CMP   #$FF
      BCC   LABEL
      RTS
```

Selbstverständlich kann auch eine andere Speicherstelle anstelle von $FB verwendet werden, sogar eine, die nicht in der Zeropage liegt. Voraussetzung ist lediglich, daß sie nicht innerhalb der Schleife verändert wird – außer zum Zählen der Schleifendurchläufe. In den bisher kennengelernten Varianten haben wir immer $FF als Endwert genommen. Nun steht man oft vor der Aufgabe, bis zu einem bestimmten Endwert zu zählen, der vorher irgendwie eingegeben oder festgelegt wird. In Basic sähe das beispielsweise so aus:

```
10 A = 32
20 FOR I = 0 TO A
30 NEXT I
```

Hier ist also der Endwert in Zeile 10 auf 32 gesetzt worden und die Schleife zählt bis zu diesem in A festgelegten Wert. In Assembler können wir das ebenfalls. Listing 2 zeigt die Variante 4.

Die Speicherstelle $FA nimmt die Funktion der Variablen A des Basicprogrammes ein. Dorthinein wird der Endwert (32=$20) gelegt und der Vergleichsbefehl lautet nun:

CPY $FA

Das ist: »Vergleiche den Inhalt des Y-Registers mit dem Inhalt der Speicherstelle $FA«. Wir haben in dieser Version 4 wieder das Y-Register als Zähler benutzt, Version 5 (Listing 3) zeigt uns dasselbe, nur wird hier die Speicherstelle $5100 zum Zählen verwendet.

Es hat sich eingebürgert, Schleifen in Assembler nicht – wie wir es bisher getan haben – aufwärts, sondern abwärts zu zählen. Der Grund dafür ist: Es geht schneller, weil man sich meistens den Compare-Befehl ersparen kann. Bei Verzögerungsschleifen ist das ja noch nicht so interessant, später aber, wenn in den Schleifen noch allerhand geschehen soll, summieren sich die Taktzeiten bei mehrfachem Durchlauf schon ganz erheblich. Eine Basic-Programmsequenz sähe nun so aus:

```
10 FOR I = 255 TO 0 STEP -1
20 NEXT I
```

In Listing 4 finden Sie das Assemblerlisting der Variante 6.

Das entspricht der Variante 1. Der Unterschied ist aber, daß hier abwärts gezählt wird und man sich den CPY-Befehl sparen kann, denn vor einem Unterlauf des Y-Registers wird automatisch bei 0 die Zero-Flagge gesetzt. Das aber prüft der BNE-Befehl.

Aus alledem ist also zu lernen:

1) Wann immer möglich, abwärts zählen.
2) Wann immer möglich, X- oder Y-Register als Zähler verwenden.

Die Variante 5 war natürlich ein ausgesuchtes Extrembeispiel, denn außer der Tatsache, daß man beim Abwärtszählen den Endwert als Startwert immer gleich in den Zähler eingeben kann und ihn normalerweise nicht noch irgendwo speichern muß, verwendet man natürlich – wenn es denn nötig ist, etwas anderes als die Indexregister dazu zu gebrauchen – eine Zeropagespeicherstelle als Zähler und nicht – wie in Version 5 – eine Speicherstelle wie $5100. Die verbesserte Version 7 entspricht dem Basicprogramm:

```
10 A = 32
20 FOR I = A TO 0 STEP -1
30 NEXTI
```

In Listing 5 finden Sie diese Version. $FA dient als Zähler.

Etwas schwieriger wird die Programmierung, wenn man nicht nur um 1 herauf- oder herunterzählt, sondern um 2, 3, 4 oder mehr. Das Basic-Äquivalent drückt sich dann beispielsweise in der Ergänzung STEP -2 der FOR...NEXT-Schleife aus. Dreht es sich nur um kleine Schrittweiten, die konstant bleiben, dann verwendet man vorteilhaft mehrere DEY (oder DEX, DEC, INY, INX und INC) hintereinander. Man muß außerdem mit der Abbruchbedingung einer solchen Schleife vorsichtig sein. BNE ist nicht immer möglich, weil man unter Umständen schon vor der Prüfung (durch

```
10   -          .LI 1,4
20   -          .BA $5000
30   -;*** VERZOEGERUNGSSCHLEIFE VERSION 8 ***
40   -;FOR I = 32 TO 0 STEP-3
50   -;
60   -          LDY #$20        ;STARTWERT IN ZAEHLER
70   -LABEL     DEY             ;MINUS 3
80   -          DEY
90   -          DEY
100  -          BPL LABEL       ;WEITER BIS UNTERLAUF
110  -          BRK
120  -;
130  -          .SY 1,4
140  -          .ST
```

Listing 6. Verzögern in kleinen Schritten mittels Variante 8

```
10   -          .LI 1,4
20   -          .BA $5000
30   -;*** VERZOEGERUNGSSCHLEIFE VERSION 9 ***
40   -;FOR I = 127 TO 0 STEP -10
50   -;
60   -          LDA #$7F        ;DAS IST DEZIMAL 127
70   -          STA $FA         ; UNSER ZAEHLER
80   -LABEL     SEC
90   -          LDA $FA
100  -          SBC #$0A
110  -          STA $FA
120  -          BPL LABEL       ;WEITER BIS UNTERLAUF
130  -          BRK
140  -          .SY 1,4
150  -          .ST
```

Listing 7. Verzögern in großen Schritten mit Variante 9

```
10   -          .LI 1,4
20   -          .BA $5000
30   -          .EQ SCREEN=$0400;BILDSCHIRMSTART
40   -          .EQ COLOR=$D800;FARBRAMSTART
50   -;*** BEISPIEL 1 ***
60   -;VERSION 8 MIT EINGFACHEM JOB
70   -;ZEICHEN AUF BILDSCHIRM ZEIGEN
80   -;
90   -;----- INITIALISIERUNG -----
100  -;
110  -          LDY #$7F        ;DAS IST DEZIMAL 127
120  -;
130  -;----- VERARBEITUNG --------
140  -;
150  -LABEL     TYA
160  -          STA SCREEN,Y
170  -          STA COLOR,Y
180  -;
190  -;----- STEUERUNG -----------
200  -;
210  -          DEY
220  -          DEY
230  -          BPL LABEL
240  -;
250  -;----- AUSGANG -------------
260  -;
270  -          BRK
280  -;
290  -          .SY 1,4
300  -          .ST
```

Listing 8. Unser Beispiel 1 in Assembler: Bunte Zeichen

BNE) unter 0 hindurch gezählt hat (dann folgt ja wieder $FF etc.). Hat beispielsweise der Zähler (hier das Y-Register) den Wert 1 und es wird durch eine Sequenz:

```
DEY
DEY
DEY
BNE LABEL
```

weitergezählt, dann nimmt Y der Reihe nach die Werte 0, FF, FE an und BNE findet die Zeroflagge nicht gesetzt. Man muß also andere Abbruchbedingungen verwenden. Solange man bis zur ersten Prüfung (also dem ersten Schleifendurchlauf beim Herunterzählen) im Zähler mindestens $7F (=binär 0111 1111) vorliegen hat, kann man mittels BPL die Schleife schließen. Zur Erinnerung: BPL verzweigt, wenn Bit 7 nicht gesetzt ist (kleiner 128), BMI verzweigt, wenn Bit 7 gesetzt ist (größer oder gleich 128). Das Basic-Programmstück

```
10 FOR I = 32 TO 0 STEP -3
20 NEXT I
```

findet seine Entsprechung in dem Assemblerlisting Version 8 (Listing 6).

Wieder dient das Y-Register als Schleifenzähler. Größere Schrittweiten lassen es – von einer gewissen Grenze an, die durch das Verhältnis von Bytezahl auf der einen und Bearbeitungsdauer auf der anderen Seite, bestimmt wird – sinnvoll erscheinen, den Zähler durch Subtraktion (oder Addition beim Aufwärtszählen) zu verändern. Das Analogon zur Basic-Sequenz:

```
10 FOR I = 127 TO 0 STEP -10
20 NEXT I
```

sehen Sie in Listing 7.

In dieser Version 9 dient die Zeropagespeicherstelle $FA als Zähler und in den Programmzeilen 80 bis 110 findet die Verminderung dieses Zählers durch Subtraktion statt ($0A = dezimal 10). Das Programm kann noch verändert werden, indem man anstelle von BPL den BCS-Befehl verwendet. Wenn die Subtraktion einen Unterlauf ergeben hat, wird das Carry-Bit gelöscht. Außerdem lassen sich noch 2 Byte einsparen, indem man das STA $FA aus Zeile 110 herausnimmt und dafür das LABEL eine Zeile höher setzt. Allerdings geht das dann auf Kosten der Durchschaubarkeit unseres Programmes.

Wir wollen nun mit den einfachen Verzögerungsschleifen aufhören. Es gäbe noch weitere Aufgaben zu lösen (beispielsweise von einem bestimmten Startwert bis zu einem bestimmten Zielwert zu zählen), die vertraue ich aber Ihnen selbst an: Alles Notwendige dazu können Sie aus unseren verschiedenen Versionen entnehmen und kombinieren. Interessant werden Schleifen hauptsächlich durch einen Job, der in ihnen wiederholt ausgeführt wird. Zwei Beispiele sollen uns zur Illustration dienen. Vorher aber sollen noch einige Bemerkungen zur grundsätzlichen Architektur von Schleifen gemacht werden.

Im Prinzip besteht jede Schleife aus vier Teilen:

- Initialisierung. Beispielsweise wird hier der Startwert des Zählers festgelegt.
- Verarbeitung. Das ist das, was in den Verzögerungsschleifen bisher leer blieb: der Job.
- Steuerung. Hoch- oder Herunterzählen des Zählers und Prüfen der Abbruchbedingung.
- Ausgang. Das war bisher bei uns immer der BRK-Befehl.

Aus diesen vier Bestandteilen lassen sich zwei grundsätzliche Schleifenmöglichkeiten konstruieren, die Sie in Bild 1 dargestellt finden.

In Bild 1a haben wir das Prinzip vorliegen, das unseren normalen FOR...NEXT-Schleifen in Basic zugrundeliegt. Diese Schleife wird mindestens einmal durchlaufen. Erst nach der Ausführung des Jobs erfolgt die Prüfung, ob die Abbruchbedingung gegeben war. Soll solch eine Schleife n-mal durchlaufen werden, muß die Initialisierung mit n-1 im Zähler erfolgen (oder die Abbruchbedingung entsprechend umgeformt werden).

Die Schleifenkonstruktion in Bild 1b dagegen muß nicht durchlaufen werden. Ihr entspricht etwa eine DO UNTIL...LOOP-Schleife oder auch eine DO WHILE...LOOP-Schleife aus dem Basic 7.0 des C 128 oder 3.5 des C16/C116. Hier erfolgt die Initialisierung des Zählers genau mit dem Wert n.

Sehen wir uns beispielsweise unsere Version 9 an, dann entdecken wir die einzelnen Schleifenteile wie folgt:

```
Initialisierung:       LDA #$7F
                       STA $FA
Verarbeitung: LABEL    ---
Steuerung:             SEC
                       LDA $FA
                       SBC #$0A
                       STA $FA
                       BCS LABEL
Ausgang:               BRK
```

Auf diese Weise ist es möglich, alle bisher kennengelernten Schleifenvarianten mit einem beliebigen Job zu füllen. Noch eins gibt es zu bedenken: Alle Instruktionen zwischen dem Label und der Abbruchbedingung werden oft ausgeführt, sind also zeitfressend. Daher sollte der auszuführende Job alle Befehle vermeiden, die ebensogut vor der eigentlichen Schleife stehen könnten.

Sehen wir uns unser 1. Beispiel an. Wir stellen uns die Aufgabe, von den 127 Zeichen, die mittels POKE-Code erfaßbar sind, jedes 2. Zeichen an jeder 2. Bildschirmstelle abzubilden. Das Ganze soll durch Verwenden verschiedener Farben auch noch hübsch bunt aussehen. In Basic würden wir dafür schreiben:

```
10 S = 1024 : C = 55296
20 FOR I = 127 TO 0 STEP -2
30 POKE S+I,I
40 POKE C+I,I
50 NEXT I
```

Weil im Bildschirmfarbspeicher nur die Bits 0 bis 3 eine Rolle spielen (die anderen aber gar nicht beachtet werden), erzeugen wir in Zeile 40 auch die verschiedenen Farben mehrmals nacheinander.

In Listing 8 wird dieselbe Problemlösung in Maschinensprache dargestellt. In den Zeilen 30, 40 und 160, 170 sehen Sie die Anwendung eines weiteren Pseudobefehls. Das .EQ bewirkt, daß eine bestimmte Speicherstelle mit einem Namen versehen werden kann. Im folgenden braucht man sich nur noch den Namen zu merken, der auch am Ende in der Symboltabelle mit ausgegeben wird. Dadurch wird man bis zu einem gewissen Grad sogar systemunabhängig. Um beispielsweise dieses Programm auf einem VC 20

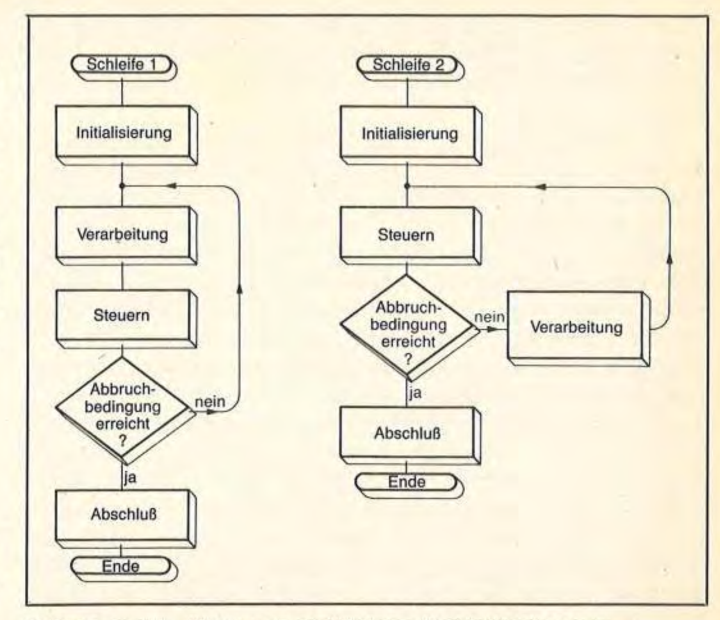


Bild 1a und b. Zwei grundsätzliche Schleifenformen

in der Grundversion laufen zu lassen, muß in Zeile 30 der SCREEN-Wert auf $1E00 und in Zeile 40 der COLOR-Wert auf $9600 geändert werden.

Bevor sie durch G 5000 aus dem Monitor heraus das Programm starten, löschen Sie am besten zuerst den Bildschirm und fahren den Cursor in eine mittlere Bildschirmzeile, damit er dem Ergebnis des Programmes nicht ins Gehege kommt. Das Programm läuft natürlich auch auf

dem C 128 (im 128er-Modus). Allerdings werden hier die Zeichen nur einfarbig, weil man zum Beschreiben des Bildschirmfarbspeichers (mit STA COLOR,Y) noch die Bank umschalten muß.

Sie sehen: Das geht in Assembler erheblich schneller als in Basic und eben die Geschwindigkeit in Assemblerprogrammen wird es sein, die uns im 2. Beispiel noch ein wenig beschäftigen wird. Die Aufgabenstellung ist folgende: Ein weißer Ball soll von rechts unten kommend quer über den Bildschirm nach links oben fliegen. Dazu sollen zwei Firmwareroutinen (auch Interpreterroutinen genannt) verwendet werden: Eine zum Drucken beliebiger Zeichen und eine andere zum Setzen des Cursors. Die erste ist das normale PRINT in Basic, das als Kernal-Routine BSOUT (manchmal auch CHROUT genannt) durch Assemblerprogramme bei $FFD2 ansteuerbar ist. Das auszudruckende Zeichen muß vor dem Aufruf JSR $FFD2 im Akkumulator enthalten sein. Die andere Routine dient dem Steuern des Cursors. Gibt man in die Speicherstelle 211 ($D3) die gewünschte Spalte und in 214 ($D6) die Zeile des Bildschirmes, an die der Cursor positioniert werden soll, dann lenkt ihn der Aufruf des bei 58640 ($E510) beginnenden Maschinenprogrammes unserer Firmware an diesen Ort.

Alle Randbedingungen werden durch dieses Basic-Programm realisiert:

```
10 S=211:Z=214:B=58640:S1=40
15 Z1=20
20 PRINT CHR$(147)CHR$(5)
30 GOSUB 100:PRINT CHR$(113)
40 FOR I=19 TO 0 STEP -1
50 GOSUB 100:PRINT CHR$(32)
60 S1=S1-2:Z1=Z1-1
70 GOSUB 100:PRINT CHR$(113)
80 NEXT I
90 PRINT CHR$(154):END
100 POKE S,S1:POKE Z,Z1:SYS B:
110 RETURN
```

In der Schleife wird immer zuerst das zuletzt gedruckte Zeichen gelöscht (sonst hätten wir nicht nur einen Ball, sondern eine Diagonale aus weißen Bällen) und dann nach dem Weitersetzen des Cursors der nächste Ball gezeichnet.

Listing 9 zeigt nun das Äquivalent dazu in Assembler. In den Zeilen 30 bis 80 finden Sie wieder den Pseudobefehl .EQ. Mit diesem werden außer den schon besprochenen Speicherstellen (Zeile, Spalte, CSET und BSOUT) auch noch zwei Zähler kreiert: COUNTZ (Zeilerzähler) und COUNTS (Spaltenzähler). Was soll das, werden Sie fragen; warum verwendet man nicht direkt ZEILE und SPALTE? Die Ursache liegt darin, daß BSOUT ebenfalls diese Speicherstellen benutzt und daher keine richtige Zählung mehr stattfinden kann. So zählt $FA und $FB und jedesmal vor Aufruf von CSET wird deren Inhalt in ZEILE und SPALTE übertragen. Wir brauchen natürlich nur einen Zähler für diese Schleife. COUNTS läuft nur nebenher und könnte auch in den Schleifenteil »Verarbeitung« geschrieben werden. Die Abbruchoperation in Zeile 480 prüft nur COUNTZ. Mehr Kommentar finden Sie direkt im Listing.

So, nun starten Sie mal das Programm nach dem Assemblieren aus dem Monitor mit G 5000! Sie meinen, da passiert ja gar nichts? Ich kann Ihnen beweisen, daß doch etwas passiert – nur so immens schnell, daß wir nichts davon sehen. Verändern Sie doch mal in Zeile 400 das #$20 (Leerzeichen) zu #$1C (Farbe rot). Das können Sie auch schnell aus dem Monitor her erreichen durch M 5033 – dort finden Sie am Anfang die 20 – und überschreiben durch 1C (RETURN drücken!). Wenn Sie nun starten, wird der Ball nicht mehr gelöscht, sondern nur rot gefärbt. Wir erhalten die Diagonale aus roten Bällen. Es geht also doch!

```
10   -          .LI 1,4
20   -          .BA $5000
30   -          .EQ SPALTE=$D3
40   -          .EQ ZEILE=$D6
50   -          .EQ COUNTZ=$FA
60   -          .EQ COUNTS=$FB
70   -          .EQ CSET=$E510
80   -          .EQ BSOUT=$FFD2
90   -;*** BEISPIEL 2 ***
100  -;BILDSCHIRMAUSGABE MIT FIRMWARE-ROUTINEN
110  -;
120  -;----- VORBEREITUNGEN -----
130  -;
140  -          LDA #$93       ;DEZIMAL 147
150  -          JSR BSOUT      ;BILDSCHIRM LOESCHEN
160  -          LDA #$05
170  -          JSR BSOUT      ;ZEICHENFARBE WEISS
180  -          LDA #$14       ;DEZIMAL 20
190  -          STA ZEILE
200  -          STA COUNTZ     ;SICHERN
210  -          LDA #$27       ;DEZIMAL 39
220  -          STA SPALTE
230  -          STA COUNTS     ;SICHERN
240  -;
250  -;----- VERARBEITUNG -------
260  -;
270  -LABEL     LDA COUNTZ
280  -          STA ZEILE
290  -          LDA COUNTS
300  -          STA SPALTE
310  -          JSR CSET       ;CURSOR SETZEN
320  -          LDA #$71       ;DEZIMAL 113
330  -          JSR BSOUT      ;GRAFIKZEICHEN DRUCKEN
340  -          NOP
350  -          LDA COUNTZ
360  -          STA ZEILE
370  -          LDA COUNTS
380  -          STA SPALTE
390  -          JSR CSET
400  -          LDA #$20       ;DEZIMAL 32
410  -          JSR BSOUT      ;ZEICHEN LOESCHEN
420  -;
430  -;----- STEUERUNG ----------
440  -;
450  -          DEC COUNTS
460  -          DEC COUNTS
470  -          DEC COUNTZ
480  -          BNE LABEL      ;HERUNTERZAEHLEN BIS 0
490  -;
500  -;----- ABSCHLUSS ----------
510  -;
520  -          LDA #$9A       ;DEZIMAL 154
530  -          JSR BSOUT      ;ZEICHENFARBE HELLBLAU
540  -          BRK
550  -
560  -          .SY 1,4
570  -          .ST
```

Listing 9. Ein schneller Flitzer: Beispiel 2

Wir müssen daher das Ganze etwas verlangsamen. Dazu ist schon eine Stelle vorgesehen: In Zeile 340 befindet sich ein gänzlich unmotiviertes NOP-Kommando. Dorthin packen wir nun eine Verzögerungsschleife und es ergibt sich das Programm in Listing 10.

In die Zeilen 335 bis 345 haben wir die Version 6, mit dem Y-Register als Zähler eingefügt. Ein erneuter Start nach dem Assemblieren zeigt uns ein kurzes weißes Aufflackern (falls Sie die Farbe Rot wieder gegen #$20 ausgetauscht haben!). Das war also immer noch zu schnell! Also bauen

```
250  -;----- VERARBEITUNG -------
260  -;
270  -LABEL     LDA COUNTZ
280  -          STA ZEILE
290  -          LDA COUNTS
300  -          STA SPALTE
310  -          JSR CSET       ;CURSOR SETZEN
320  -          LDA #$71       ;DEZIMAL 113
330  -          JSR BSOUT      ;GRAFIKZEICHEN DRUCKEN
335  -          LDY #$FF       ;VERZOEGERUNG
340  -MARKE     DEY
345  -          BNE MARKE
350  -          LDA COUNTZ
360  -          STA ZEILE
370  -          LDA COUNTS
380  -          STA SPALTE
390  -          JSR CSET
400  -          LDA #$20       ;DEZIMAL 32
410  -          JSR BSOUT      ;ZEICHEN LOESCHEN
420  -;
430  -;----- STEUERUNG ----------
```

Listing 10. Flitzer mit kleinem Handicap

wir noch eine Verzögerungsschleife ein (Zeilen 346 bis 348 in Listing 11).

Nun sehen wir schon ein wenig mehr, aber wir können uns vorstellen, daß es reichlich ungelenk wäre, nun noch eine dritte, vierte,... Verzögerung einzubauen. Es gibt noch einen anderen Weg, nämlich einfach 2 Verzögerungen ineinander zu verschachteln. Das ist schließlich in Listing 12 geschehen und wenn Sie das nach der Assemblierung starten, dann geht's hübsch langsam. Immerhin wird die innere Schleife 255*255mal durchlaufen. Jedesmal nämlich, wenn wir X bis 0 heruntergezählt haben, wird Y dekrementiert und X wieder mit #$FF beladen. Das geht so lange, bis auch Y auf 0 heruntergezählt wurde. Wenn Sie in Zeile 332 statt #$FF einen kleineren Startwert eingeben (geht wieder ganz gut vom Monitor aus), läuft der Ball schneller. Damit haben Sie die Geschwindigkeit völlig im Griff.

Wir haben jetzt die einfachen 8-Bit-Schleifen verlassen, denn diese Verzögerung ist schon eine 16-Bit-Schleife.

16-Bit-Schleifen

Sehen wir uns zunächst einmal in Basic an, was wir da gemacht haben. Es dreht sich um etwas uns sehr Bekanntes: Zwei ineinander geschachtelte Schleifen. Am genauesten entspricht wohl diese Programmsequenz unserer 16-Bit-Verzögerung:

```
100    Y=255
110    X=255
120    X=X-1
130    IF X .. 0 THEN 120
140    Y=Y-1
150    IF Y .. 0 THEN 110
```

Gebräuchlicher wäre allerdings diese Version:

```
100    FOR Y=255 TO 0 STEP-1
110    FOR X=255 TO 0 STEP-1
120    NEXT X
130    NEXT Y
```

Dagegen halten wir unsere Verzögerungsschleife aus dem letzten Assemblerprogramm (Listing 12):

```
       LDY #$FF
LABEL  LDX #$FF
MARKE  DEX
       BNE MARKE
       DEY
       BNE LABEL
       ...
```

Diese Schleife zählt das X-Register so oft eine ganze Page (minus 1, also jeweils 255mal) durch, wie es das Y-Register angibt, hier also 255mal. Insgesamt finden daher 255*255 = 65025 Durchläufe statt. Um ganze Pages, also 256 Zählungen zu erreichen, lädt man ins X-Register einfach 0 ein. Der DEX-Befehl sorgt dann noch vor der BNE-Prüfung für einen Unterlauf auf $FF.

Deutlich wird Ihnen sicher, daß wir – im Gegensatz zur einfachen Schleife – hier einen Multiplikationseffekt zu beachten haben. Die Anzahl der Durchläufe setzt sich zusammen aus:

Y-Startwert * X-Startwert

Das ist auch ganz akzeptabel, solange man die gewünschte Durchlaufzahl aus zwei Faktoren zusammensetzen kann. Soll ein Job beispielsweise 1000mal ausgeführt werden, dann gibt es mehrere Möglichkeiten, denn

```
1000 = 8 * 125
     = 4 * 250
     =10 * 100
```

Wir könnten dann unsere Job-Schleife schreiben:

```
        LDY #$04
LABEL   LDX #$FA
MARKE   Job-Befehle
        DEX
        BNE MARKE
        DEY
        BNE LABEL
        ...
```

Abgesehen davon, daß es doch ein wenig aufwendig ist – besonders bei einer nicht festgelegten Anzahl von Durchläufen – jedesmal eine Aufspaltung in zwei Faktoren vorzunehmen: Was tun wir bei Primzahlen? 997 Jobs beispielsweise lassen sich in solch einer Doppelschleife nicht bearbeiten (997 ist eine Primzahl, das bedeutet, diese Zahl ist nicht in Faktoren zerlegbar).

Im Prinzip gibt es für solche Fälle zwei Lösungen:

– Entweder stellt man fest, daß es gleichgültig ist, ob nun (um bei unseren Beispielen zu bleiben) 1000, 1024 oder 997 Durchläufe stattfinden. Es ist häufig der Fall, daß dadurch nicht mehr Schaden angerichtet wird als der zusätzliche Zeitbedarf für 27 Durchläufe (bei 1024 anstelle von 997). In diesem Fall legt man den Anfangswert der inneren Schleife einfach grundsätzlich auf 0 fest (arbeitet also genau eine Page darin ab) und variiert nach Bedarf den Startwert der äußeren Schleife (dort wird nun also 4 eingetragen).

– Oder aber – wenn's genau drauf ankommt – wir müssen zwei Schleifen einrichten: Für die ganzen Pages eine Doppelschleife und für den Rest eine einfache. Genau das

```
250  -;----- VERARBEITUNG -------
260  -;
270  -LABEL      LDA COUNTZ
280  -          STA ZEILE
290  -          LDA COUNTS
300  -          STA SPALTE
310  -          JSR CSET        ;CURSOR SETZEN
320  -          LDA #$71        ;DEZIMAL 113
330  -          JSR BSOUT       ;GRAFIKZEICHEN DRUCKEN
335  -          LDY #$FF        ;VERZOEGERUNG
340  -MARKE      DEY
345  -          BNE MARKE
346  -          LDY #$FF
347  -WEITER     DEY
348  -          BNE WEITER
350  -          LDA COUNTZ
360  -          STA ZEILE
370  -          LDA COUNTS
380  -          STA SPALTE
390  -          JSR CSET
400  -          LDA #$20        ;DEZIMAL 32
410  -          JSR BSOUT       ;ZEICHEN LOESCHEN
420  -;
430  -;----- STEUERUNG ----------
```

Listing 11. Der doppelt zögernde Flitzer

```
250  -;----- VERARBEITUNG -------
260  -;
270  -LABEL      LDA COUNTZ
280  -          STA ZEILE
290  -          LDA COUNTS
300  -          STA SPALTE
310  -          JSR CSET        ;CURSOR SETZEN
320  -          LDA #$71        ;DEZIMAL 113
330  -          JSR BSOUT       ;GRAFIKZEICHEN DRUCKEN
332  -          LDY #$FF
334  -MARKE      LDX #$FF
336  -WEITER     DEX
338  -          BNE WEITER
340  -          DEY
342  -          BNE MARKE
350  -          LDA COUNTZ
360  -          STA ZEILE
370  -          LDA COUNTS
380  -          STA SPALTE
390  -          JSR CSET
400  -          LDA #$20        ;DEZIMAL 32
410  -          JSR BSOUT       ;ZEICHEN LOESCHEN
420  -;
430  -;----- STEUERUNG ----------
```

Listing 12. Der Flitzer ist voll unter Kontrolle

geschieht in einer sehr nützlichen Routine unserer Firmware, der BLTUC- (oder auch Blockverschiebe-) Routine, auf die wir später genauer eingehen wollen. Sie können ja schon mal mittels SMON in den Speicher sehen: Von $A3BF bis A3FA ist dieses Programm zu finden.

Bevor wir uns an diese schwierigeren Sachen wagen, wollen wir uns aber noch ein wenig mit Fragen der Schleifenstruktur befassen. Zunächst kann nur relativ selten auf die beiden Indexregister als Zähler zurückgegriffen werden. Man muß meistens zwei Speicherstellen dazu verwenden. Außerdem kann man natürlich ebensogut in den Schleifen aufwärts zählen. Das soll im folgenden Beispiel beides geschehen, wo wir den Bildschirminhalt invertieren wollen. Das geschieht einfach durch Setzen des Bit 7 des Codes in jeder Bildschirmspeicherstelle (wir machen das durch EOR $80). Das hat den Vorzug, daß ein zweiter Durchlauf des Programmes wieder den Ausgangszustand des Bildschirmes herstellt. Zuerst sollen Sie eine etwas schwerfällige, aber überschaubare Form des Programmes kennenlernen (Listing 13):

Hier wurden – auf höchst plumpe Weise – vier ganze Pages bearbeitet. Eine andere Lösung wäre es, anstelle von $FA in der Zeile 4010 das Y-Register zu erhöhen (mittels INY). Es würde dann sowohl als Index als auch als Zähler dienen. (In unserer Version hatte es ja nur eine Alibifunktion für die spezielle Art der Adressierung der Bildschirmspeicherzellen). Eleganter kann das Problem gelöst werden mit einer Technik, die Florian Müller in seinem Artikel »Effektives Programmieren in Assembler« (64'er Sonderheft 8, 1985, S.22) vorstellt. Dabei werden $FA und $FB nicht mehr als Zähler verwendet, sondern dem Y-Register kommt wieder die Doppelfunktion zu als Index und als Zähler der inneren Schleife. Das X-Register ist Zähler der äußeren Schleife. In der inneren wird Y hoch-, in der äußeren Schleife X heruntergezählt. Das Ergebnis davon ist: Das Programm wird kürzer und auch schneller (Listing 14).

Es stört uns manchmal immer noch, daß wir – statt nur bis $07E7 (denn das ist dezimal 2023) – bis $07FF invertieren. Bevor wir in der nächsten Folge die oben erwähnte Variante ergründen, die in der BLTUC-Routine verwendet wird, soll Ihnen noch eine weitere Möglichkeit vorgestellt werden, die im SMON und neuerdings auch von F. Müller (siehe oben) gezeigt worden ist. Da geht's recht trickreich zu.

Wieder wird pro forma das Indexregister Y initialisiert wegen der speziellen Art der Adressierung (Listing 15).

Natürlich wird diese Doppelschleife durch die ständigen Rechnungen im Steuerteil relativ langsam, weshalb es doch lohnt, auch andere Wege zu untersuchen.

Zwei ROM-Routinen

Kommen wir – wie versprochen – noch auf die beiden vorhin verwendeten Routinen zurück, die sich im oberen ROM-Bereich unseres Computers befinden. Die eine davon ($FFD2) ist mittlerweile schon vielen recht geläufig. Sie dient dazu, ein im Akku enthaltenes Zeichen an ein vorher definiertes Gerät auszugeben. Der Unterschied zwischen beiden Routinen ist, daß CHROUT (also $FFD2) sich im sogenannten Kernel-Bereich befindet, die andere (PLOTK $E510) aber nicht. Was ist denn nun das Besondere am Kernel-Bereich? Es handelt sich um eine Tabelle von 39 JMP-Befehlen, für die Commodore garantiert, daß sie in allen Computerversionen an der gleichen Stelle liegt und gleiche Funktionen beinhaltet. Leider existiert diese Möglichkeit des Kernel nur für relativ wenige Verwendungszwecke. Wer beispielsweise Fließkommaoperationen in Assembler zu programmieren hat, sucht oft ziemlich verzweifelt im ROM eines neuen Computers nach den dazu

```
Initialisierung:
 4000    LDA #$00       Die Bildschirmadresse wird
 4002    STA $FA        in den Vektor $FA/FB geschrieben.
 4004    LDA #$04       Index auf Null.
 4006    STA $FB
 4008    LDY #$00
Job:
 400A    LDA ($FA),Y    Code in Akku
 400C    EOR #$80       invertieren und
 400E    STA ($FA),Y    zurückschreiben.
Steuerung:
 4010    INC $FA        LSB hochzählen
 4012    BNE $400A      und weiter Job ausführen, bis ein Über-
                        lauf von 255 auf 0 stattfindet.
 4014    INC $FB        dann MSB erhöhen
 4016    LDA $FB        und prüfen, ob
 4018    CMP #$08       Endadresse erreicht ist.
 401A    BNE $400A      Falls noch nicht, erneut zur
                        Jobschleife
Ausgang:
 401C    BRK            Sonst Ende mit Registeranzeige.
```

Listing 13. Invertieren des Bildschirms

```
Initialisieren
 4000    LDA #$00       LSB Bildschirm
 4002    STA $FA        in Vektor und
 4004    TAY            Index = 0.
 4005    LDA #$04       MSB in Vektor
 4007    STA $FB        schreiben und
 4009    TAX            Zähler für die pages auf 4.
Job:
 400A    LDA ($FA),Y    Dasselbe wie
 400C    EOR #$80       wir es vorhin
 400E    STA ($FA),Y    hatten.
Steuerteil:
 4010    INY            Index (Zähler)+1
 4011    BNE $400A      wenn noch kein Überlauf, erneut Job aus-
                        führen.
 4013    DEX            sonst page-Zähler herunterzählen.
 4016    BNE $400A      Wenn noch nicht 0, dann wieder Jobbear-
                        beitung.
Ausgang:
 4018    BRK            sonst wieder Ende mit Registeranzeige.
```

Listing 14. Verbesserte Form von Listing 6

```
Initialisieren:
 4000 LDA #$00          Bildschirmstart
 4002 STA $FA           in Vektor $FA/FB
 4004 LDA #$04
 4006 STA $FB
 4008 LDY #$00

Job:
 400A    LDA ($FA),Y    Das kennen wir
 400C    EOR #$80       nun schon.
 400E    STA ($FA),Y
Steuerung:
 4010    INC $FA        Erhöhen des LSB
 4012    BNE 4016       Wenn kein Überlauf, erfolgt ein Sprung.
 4014    INC $FB        Sonst auch Erhöhen des MSB.
 4016    LDA $FA        Das LSB wird nun
 4018    CMP #$E8       verglichen mit dem MSB der Endadresse +
                        1. Dabei findet die Resultatanzeige in
                        den Flaggen (N,Z,C) statt.
 401A    LDA $FB        Nun wird das MSB der Adresse in den Akku
                        geladen und
 401C    SBC #$07       das MSB der Endadresse subtrahiert. Die
                        Carryflaggge ist gesetzt, wenn die Adres-
                        se in $FA/FB gleich der Endadresse+1
                        ($07E8) geworden ist.
 401E    BCC 400A       Solange das noch nicht der Fall ist, wird
                        zum Job zurückverzweigt.
Ausgang:
 4020    BRK            Sonst aber Ende mit Registeranzeige.
```

Listing 15. Die trickreichste Version

passenden Routinen. Hier hilft meist nur die Anschaffung eines ROM-Listings.

Alle Kernel-Routinen verlangen eine festgelegte Bearbeitungsweise:
a) Vorbereitungen treffen
b) Routinenaufruf
c) Fehlerabfrage und -behandlung (einige Fehlermeldungen finden Sie in Tabelle 1)

Damit hätten wir die Vorrede hinter uns und können uns dem CHROUT-Programm zuwenden, das wir an dieser Stelle in seiner eingeschränkten Funktion betrachten, nämlich zur Ausgabe des Akku-Inhaltes auf dem Bildschirm. Falls Sie eine detaillierte Schilderung weiterer Anwendungsmöglichkeiten suchen sollten: Im Assembler-Kurs (64'er-Sonderheft, Ausgabe 8/85, Seite 33 und ab Seite 39) finden Sie beispielsweise die Ausgabe auf den Drucker.

```
Name CHROUT (auch BSOUT)
Zweck                      Ausgabe eines Zeichens
Adresse                    $FFD2, 65490
Vorbereitungen             (CHKOUT,OPEN)
Parameter Eingabeort       Akku
Eing.Format                ASCII
Ausgabeort                 spezifiziertes Gerät
Ausg.format                -
Fehler                     0
Stapelbedarf               8
Register                   Akku
```

CHROUT ist freundlicherweise so geschaffen worden, daß von den Vorbereitungen lediglich übrigbleibt das Zeichen in den Akku zu bringen, falls man nur die Bildschirmausgabe wünscht. CHROUT ist zwar ein enorm vielseitiger, aber leider auch etwas langsamer Geselle. Das liegt daran, daß CHROUT gewissermaßen als die eierlegende Wollmilchsau konstruiert wurde, also fast alles kann. Damit sind aber endlos viele Prüfungen und Abfragen verbunden, die man sich durch Verwenden anderer Routinen - die lernen Sie noch kennen - ersparen kann.

Nun zur zweiten Adresse $E510, der PLOTK-Routine.

```
Name PLOTK
Zweck            Cursor setzen
Adresse          $E510, 58640
Vorbereitungen   Zeile in 214, Spalte in 211
Parameter        Übergaben spielen hier keine
                 Rolle.
Fehler           spielen nur bei Kernel-Routinen
                 eine Rolle.
Stapelbedarf     2
Register         Akku, X, Y
Weitere Speicherstellen, die durch die Routine
beschrieben werden können: 209, 210, 213, 217 (alle
als Dezimalzahlen).
```

Dies ist nur eine der möglichen Einsprungadressen dazu. Es handelt sich nicht um eine Kernel-Routine: Prompt findet sich auch in dem dazugehörigen Programm an einer anderen Einsprungstelle ein Unterschied bei verschiedenen C 64-ROMs, der uns aber nicht zu kümmern braucht.

Diese letzte Angabe werden Sie nicht bei allen beschriebenen Routinen finden. Manchmal ist der Irrweg, dem man durch das ROM zu folgen hat, so komplex, daß ich Ihnen empfehle, selbst mal per SMON (Trace-Kommandos) durchs Labyrinth zu gehen.

Vier Anwendungen der Schleifenprogrammierung werden Sie im folgenden kennenlernen: Das Löschen des Grafikspeichers, das Beschreiben des Grafik-Farbspeichers, einen Rahmenaufbau um den Bildschirm und schließlich das Beschreiben eines Bildschirmfensters.

Nr.	Text	Bedeutung
0	BREAK	Während des Programmes wurde die RUN/STOP-Taste gedrückt
1	TOO MANY FILES	Man kann maximal 10 offene Files einrichten
2	FILE OPEN	Ein bereits geöffneter File wird nochmal geöffnet
3	FILE NOT OPEN	Auf einen noch nicht geöffneten File sollte zugegriffen werden
4	FILE NOT FOUND	Der geforderte File ist nicht verfügbar
5	DEVICE NOT PRESENT	Das angesprochene Gerät zeigt keine Reaktion
6	NOT INPUT FILE	Aus einem Schreibfile kann nicht gelesen werden
7	NOT OUTPUT FILE	In einem Lesefile kann nicht geschrieben werden
8	MISSING FILE NAME	Bei Operationen, die einen Filenamen erfordern, fehlt dieser
9	ILLEGAL DEVICE NUMBER	Das versuchte Kommando ist beim angesprochenen Gerät nicht möglich

Tabelle 1. Fehlernummern und ihre Bedeutung. Die Nummern findet man nach Aufruf von Kernel-Routinen bei gesetztem Carry im Akku.

Dies ist gleichzeitig auch die erste Reaktion auf Ihre Fragen, die Sie uns zum Thema Assembler gesandt haben. Noch eine technische Vorbemerkung: Alle nun folgenden Programme wurden mit einem Nachfolger des Hypra-Ass - dem Programm Top-Ass - geschrieben. Aus dem Top-Ass wurden aber nur die Optionen verwendet, die auch in Hypra-Ass enthalten sind. Einige Pseudo-Opcodes heißen etwas anders, die Bedeutung ist aber leicht zu erkennen. In den Kommentaren finden Sie den jeweiligen Befehl auch in der Hypra-Ass-Syntax.

Oben hatten wir uns schon eine spezielle Form der Doppelschleife angesehen, die es möglich machte, auch von ganzen Seiten (Pages) abweichende Speicherbereiche zu bearbeiten. Als Beispiel hatten wir den Bildschirminhalt invertiert. Diese Doppelschleife soll in verallgemeinerter Form diesmal Verwendung finden. Bild 2 zeigt Ihnen ein Flußdiagramm dieser allgemeingültigen 16-Bit-Schleife.

Schon in dem Beispiel zur Invertierung hatten wir die Startadresse als Vektor in zwei Zeropage-Speicherstellen geschrieben. Nun wird auch die Endadresse als Vektor $FC/FD gespeichert. So braucht nur im Initialisierungsteil von Aufgabe zu Aufgabe eine Änderung vorgenommen zu werden.

Noch allgemeiner kann die Doppelschleife gestaltet werden durch eine Änderung des Jobteils. Handelt es sich beispielsweise um die Aufgabe, bestimmte Speicherbereiche zu beschreiben, dann kann auch der einzuschreibende Wert in eine Zeropage-Speicherstelle gepackt werden (hier in $FE). Will man allerdings auch die Art des Jobs offenhalten, dann verwendet man lediglich Sprünge in Unterprogramme. Der Jobteil heißt dann nur noch:

JSR JOB

Unsere Aufgabe ist es dann, an der Stelle JOB jeweils das gebrauchte Unterprogramm bereitzuhalten. Allerdings sollten solche Schleifenformen nicht oft benutzt werden, denn die Sprünge ins Unterprogramm verbrauchen relativ viel Rechenzeit.

Zwei Fragen treten häufig auf, die das Löschen oder Neubeschreiben der Grafikspeicher betreffen. Beide sind mit ein und derselben Doppelschleife lösbar. Sehen wir uns einmal die Sache mit dem Grafik-Farbspeicher an.

Grafik-Farbspeicher belegen

Im allgemeinen verwendet ein C 64-Grafik-Programmierer eine Bit-Map, die bei 8192 startet und ein Farb-RAM, das anstelle des normalen Bildschirmes (also ab 1024 = $400) zu finden ist. Der Benutzer des C 128 hat die Bit-Map am gleichen Ort, aber dafür ein extra Grafik-Farb-RAM ab $1C00. Zwar hat das Basic 7.0 des C 128 allerlei nette Grafik-Befehle anzubieten: Wenn aber gewünscht wird, eine schon auf dem Bildschirm sichtbare Zeichnung mit anderen Farben zu zeigen, stehen beide (also C 64- und C 128-Benutzer) vor demselben Problem. Eine globale Farbänderung ist beim C 128 nur bei gleichzeitigem Löschen der Bit-Map möglich! Wie ist die Aufgabe zu lösen?

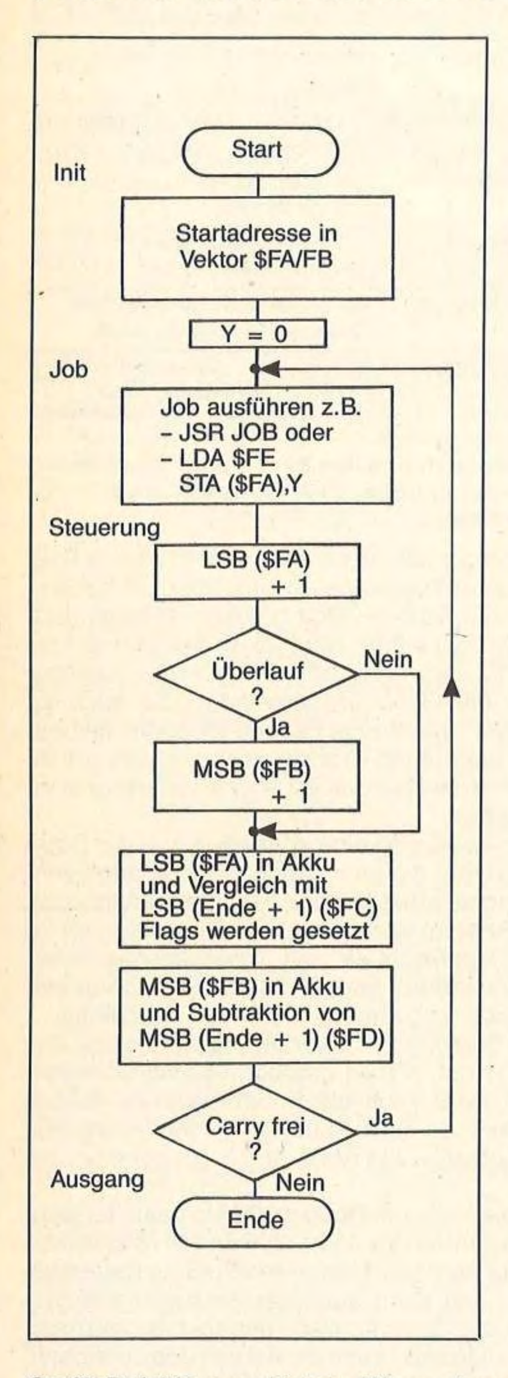


Bild 2. Flußdiagramm einer Doppelschleife für beliebige Laufzahlen

```
10   -.list 1,4,7                ;in hypra-ass: .li 1,4,7
20   -.base $c000                ;in hypra-ass: .ba $c000
30   -;**********************************************
40   -; 16-bitschleife anwendung : screen-speicher
50   -;**********************************************
60   -;
70   -.define start    = $0400 ;in hypra-ass statt .define
80   -.define ende     = $07e8 ;jeweils: .eq  zum beispiel
90   -.define wert     = $f0   ; .eq start = $0400
100  -;
110  -;---------- initialisierung ----------
120  -;
130  -          lda #<(start)  ;lsb startadresse
140  -          ldy #>(start)  ;msb startadresse
150  -          sta $fa        ;in vektor $fa/fb schreiben
160  -          sty $fb
170  -;
180  -          lda #<(ende)   ;lsb endadresse+1
190  -          ldy #>(ende)   ;msb endadresse
200  -          sta $fc        ;in vektor $fc/fd schreiben
210  -          sty $fd
220  -;
230  -          lda #wert      ;einzuschreibenden wert
240  -          sta $fe        ;nach $fe schreiben
250  -;
260  -          ldy #$00       ;index auf null stellen
270  -;
280  -;---------- job ausfuehren ------------
290  -;
300  -label     lda $fe        ;wert laden
310  -          sta ($fa),y    ;und eintragen
320  -;
330  -;--------- steuerteil -----------------
340  -;
350  -          inc $fa        ;lsb start nun als zaehler erhoehen
360  -          bne marke      ;falls kein ueberlauf weiter
370  -          inc $fb        ;sonst msb ebenfalls erhoehen
380  -marke     lda $fa        ;vergleich des lsb
390  -          cmp $fc        ;mit lsb der endadresse (flaggen setzen)
400  -          lda $fb        ;vom msb des zaehlers
410  -          sbc $fd        ;wird das msb der endadresse subtrahiert
420  -          bcc label      ;zurueck zum job wenn zaehler < endadresse
430  -;
440  -;-------- ausgang --------------------
450  -;
460  -          brk            ;sonst programmende
470  -;
480  -.symbols u,1,4,7          ;in hypra-ass: .sy 1,4,7
```

Listing 16. Ein Programm zum Beschreiben des Grafik-Farb-RAM

```
10   -.list 1,4,7                ;in hypra-ass: .li 1,4,7
20   -.base $c000                ;in hypra-ass: .ba $c000
30   -;**********************************************
40   -; 16-bitschleife anwendung : bitmap-loeschen
50   -;**********************************************
60   -;
70   -.define start    = $2000 ;in hypra-ass: .eq start = $2000
80   -.define ende     = $3f3f ;   -"-        .eq ende  = $3f3f
90   -.define wert     = $00   ;   -"-         .eq wert = $00
100  -;
110  -;---------- initialisierung ----------
120  -;
130  -          lda #<(start)  ;lsb startadresse
140  -          ldy #>(start)  ;msb startadresse
150  -          sta $fa        ;in vektor $fa/fb schreiben
160  -          sty $fb
170  -;
180  -          lda #<(ende)   ;lsb endadresse+1
190  -          ldy #>(ende)   ;msb endadresse
200  -          sta $fc        ;in vektor $fc/fd schreiben
210  -          sty $fd
220  -;
230  -          lda #wert      ;einzuschreibenden wert
240  -          sta $fe        ;nach $fe schreiben
250  -;
260  -          ldy #$00       ;index auf null stellen
270  -;
280  -;---------- job ausfuehren ------------
290  -;
300  -label     lda $fe        ;wert laden
310  -          sta ($fa),y    ;und eintragen
320  -;
330  -;--------- steuerteil -----------------
340  -;
350  -          inc $fa        ;lsb start nun als zaehler erhoehen
360  -          bne marke      ;falls kein ueberlauf weiter
370  -          inc $fb        ;sonst msb ebenfalls erhoehen
380  -marke     lda $fa        ;vergleich des lsb
390  -          cmp $fc        ;mit lsb der endadresse (flaggen setzen)
400  -          lda $fb        ;vom msb des zaehlers
410  -          sbc $fd        ;wird das msb der endadresse subtrahiert
420  -          bcc label      ;zurueck zum job wenn zaehler < endadresse
430  -;
440  -;-------- ausgang --------------------
450  -;
460  -          brk            ;sonst programmende
470  -;
480  -.symbols u,1,4,7          ;in hypra-ass: .sy 1,4,7
```

Listing 17. Blitzartiges Löschen der Bit-Map

In beiden Fällen ist in 1000 Speicherstellen ein bestimmter Wert einzuschreiben: Beim C 64 von $400 bis $7E8, beim C 128 von $01C00 bis $01FE8. Der Farbcode, der einzutragen ist, hat in beiden Fällen denselben Aufbau: Das untere Nibble (also die Bits 0 bis 3) enthält den Code der Hintergrund-, das obere Nibble (Bits 4 bis 7) den der Zeichenfarbe. C 128-Benutzer müssen vom Farbcode jeweils eine 1 abziehen. Sei ZF die Zeichen- und HF die Hintergrundfarbe, dann folgt für den einzuschreibenden Code F:

$$F = 16 * ZF + HF$$

Als Listing 16 finden Sie eine mögliche Lösung des Problems. Hier wurde in der Initialisierung (Zeilen 110 bis 270) auch die Belegung der Zeropage-Adressen mit dem Startwert ($FA/FB), dem Endwert ($FC/FD) und dem Farbcode ($FE) vorgenommen:

Zum Listing muß sicher nichts mehr gesagt werden: Es ist ausführlich kommentiert und entspricht der oben im

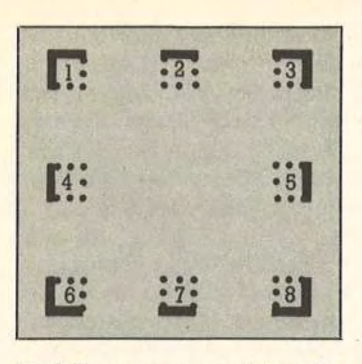


Bild 3. Zum Rahmenproblem: Diese Zeichen benötigen Sie, wenn Sie einen Bildschirmrahmen konstruieren wollen.

Flußdiagramm gezeigten Doppelschleife. Zum Starten dieses Maschinenprogrammes: Wenn Sie es einfach durch

```
SYS 49152(C 128: BANK 0: SYS49152)
```

mit auf dem Bildschirm vorhandener Abbildung ablaufen lassen, wird der Farbcode $F0 eingetragen, also hellgrau auf schwarzen Hintergrund. Falls Sie im Textmodus sind, entspricht das dem Zeichen mit dem Code $F0 (dezimal 240, ein inverses Grafik-Zeichen), das nun den gesamten Bildschirm ziert. Möchten Sie eine andere Farbenkombination erzielen, dann haben Sie mehrere Möglichkeiten. Entweder lassen Sie einfach die Zeilen 230 und 240 wegfallen und POKEn vor dem Routinenaufruf Ihren Wert F in die Speicherstelle $FE oder aber Sie verändern das Maschinenprogramm vor dem Aufruf, indem Sie an die Stelle in Zeile 230, an der der WERT steht, Ihr F POKEn. Das ist (vorausgesetzt, Sie belassen den Start bei $C000) die Speicherzelle $C011 (das ist dezimal 49169). Ein Aufruf könnte dann beispielsweise so aussehen:

```
10 INPUT"ZF,HF =";ZF,HF
20 F = 16*ZF + HF
30 POKE 49169,F
40 SYS 49152
```

Auf ähnliche Weise können natürlich auch die Start- und/oder Endadressen variiert werden, so daß beispielsweise nur der halbe Farbspeicher neu belegt wird. C128-Benutzer müssen zum Ändern des Grafik-Farb-RAM die Werte in den Zeilen 70 und 80 auf die oben genannten Adressen einstellen.

Bit-Map löschen

Vom eben gezeigten Beispiel zum Löschen einer Bit-Map ist es nur ein kleiner Schritt. Lediglich der Ort einer Bit-Map ist ein anderer und ihr Umfang. Im allgemeinen startet der Grafik-Speicher bei 8192 ($2000), hat eine Ausdehnung von 8000 Bytes (nämlich 200*320/8) und endet bei $3F3F (dezimal 16191). Der einzutragende Wert ist Null. Listing 16 unterscheidet sich vom Listing 17 somit nur durch die Zeilen 70 bis 90.

Daran können Sie ersehen, wie vielseitig unsere Doppelschleife ist. C 128-Benutzer werden meinen, daß sie das Listing 17 und – falls Ihnen niemals das Problem der Farbänderung eines fertigen Bildes unterkommt – auch das Listing 16 entbehren könnten. Sollten Sie aber jemals in die Lage kommen, eine andere Bit-Map und einen anderen Grafik-Farbspeicher als die softwaremäßig voreingestellten bedienen zu wollen, dann sind Sie in der gleichen Lage wie ein C64-Benutzer. Alle Grafik-Kommandos zielen nur auf die oben erwähnten Standard-Grafik-Speicherbereiche. Jede andere Bit-Map und jedes andere Farb-RAM muß mit selbstgebastelten Befehlssequenzen behandelt werden.

Noch eine kleine Bemerkung am Rande: Falls es Ihnen in den Sinn kommen sollte, das zum Listing 17 gehörende Maschinenprogramm einfach mal zu starten, dann bedenken Sie, daß durch Hypra-Ass im fraglichen – zu löschenden – Speicherbereich Quelltexte und Label abgelegt sind. Speichern Sie diese also vor dem Start ab! Noch schlimmer ergeht es C 128-Benutzern, die den Top-Ass verwenden: Das Löschen des Grafik-Bereiches reißt ein tiefes Loch ins Top-Ass-Programm. Top-Ass und normale Grafik können nicht gleichzeitig betrieben werden.

Bildschirm umrahmen

Viele Leser wollen wissen, wie man in Assembler einen Rahmen um den Textbildschirm legen kann. Gehen wir also dieses Problem möglichst allgemeingültig an. Wir brauchen dazu acht verschiedene Grafikzeichen (siehe dazu das Bild 3).

Außerdem erkennen wir recht schnell, daß wir bei diesem Problem andere Schleifen benutzen müssen. In Bild 4 ist die Aufgabe grafisch aufgeschlüsselt.

Die Nummern entsprechen jeweils den verschiedenen Zeichen, die in eine Speicherstelle einzuschreiben sind. Es handelt sich um folgende:

Nr.	Hex.	Dez.	Beschreibung
1	4F	79	Winkel links oben
2	77	119	Linie oben
3	50	80	Winkel rechts oben
4	74	116	Linie linksseitig
5	6A	106	Linie rechtsseitig
6	4C	76	Winkel links unten
7	6F	111	Linie unten
8	7A	122	Winkel rechts unten

Im Listing 18 sind diese acht Zeichen in der Initialisierungsphase in acht Zeropage-Speicherstellen geschrieben worden ($1B bis $22), die CODE1 bis CODE8 genannt wurden.

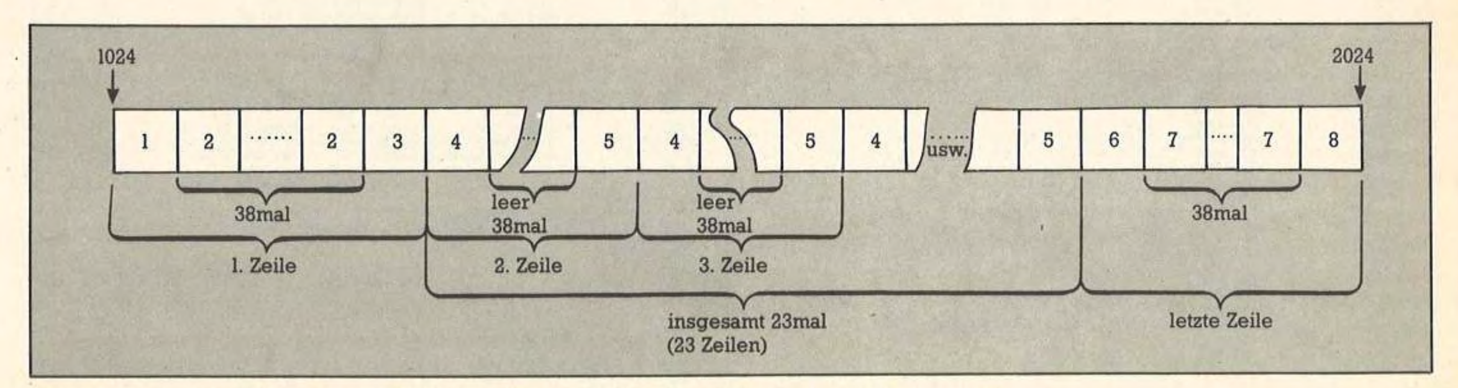


Bild 4. Zur Verdeutlichung von Bild 3 hier die grafische Darstellung des Rahmenproblems

Bevor wir dieses Programm besprechen, gebe ich Ihnen in Bild 5 noch das Flußdiagramm dazu an.

Das Programm ist in drei Teilen angeordnet: Die Initialisierung, in der alle Startwerte festgelegt werden. Sie liegen durchweg in Zeropage-Speicherstellen, was durch einfaches Ändern dieses Programmteils das Programm flexibel hält. Wir hätten noch einen Schritt weiter gehen können, indem wir auch die einzuschreibenden Werte mit .define (oder .eq bei Hypra-Ass) definiert hätten. Aber ein wenig sollen Sie auch noch zu tun behalten. Als Phase 2 tritt dann die eigentliche Bearbeitung auf, die wiederum in drei Abschnitte unterteilbar ist: obere Zeile, Mittelteil und untere Zeile. Phase 3 ist der Abschluß des Programmes, der hier nur mittels eines BRK vollzogen wurde. Ein RTS erlaubt Ihnen die Benutzung auch von Basic aus.

Sehen wir uns nun die Initialisierung im einzelnen an (Zeilen 470 bis 780): Es gibt immer noch eine ganze Menge C 64-Modelle, bei denen ein einfacher POKE-Befehl in den Bildschirmspeicher nicht ausreicht, ein Zeichen sichtbar werden zu lassen. Dann muß nämlich an der dazugehörigen Bildschirmfarbspeicherstelle ein Farbcode enthalten sein. Deshalb legen wir gleich zu Beginn einen Code 2 (für die Farbe Rot) ins X-Register, der dann später parallel zum Zeichencode in den Farbspeicher ab $D800 geschrieben wird. Es folgt ab Zeile 500 die schon erwähnte Eintragung der Zeichencodes in acht Zeropage-Speicherstellen ($1B bis $22, genannt CODE1 bis CODE8). Ab Zeile 660 werden zwei Vektoren erzeugt: $FB/FC enthält die Startadresse des Bildschirmes, $FD/FE die des Farb-RAM. Diese Vektoren heißen SCREEN und COLOR. Ab Zeile 740 werden zwei Schleifengrenzen festgelegt: in HORIZ schreiben wir die maximale Spaltenzahl pro Zeile (0...39) und nach VERT kommt eine Zahl, die die Menge der Zeilen (25 – Kopf- und Fußzeile) enthält. Schleifenzähler ist das Y-Register, das in Zeile 780 auf Null gesetzt wird.

Damit sind alle wichtigen Parameter in Zeropage-Speicherzellen untergebracht und wir können im folgenden Hauptteil des Programmes durchgängig die zeitlich günstige Form der Zeropage-Adressierung verwenden.

Ab Zeile 800 folgt nun der Rahmenaufbau. Wir beginnen mit der oberen Zeile. Zuerst wird in Speicherstelle 1024 das erste Zeichen eingetragen. Dann verschieben wir den Farbcode aus dem X-Register in den Akku und schreiben ihn in den Farbspeicher. Die Sequenz

```
STA (SCREEN),Y
TXA
STA (COLOR),Y
INY
```

wird im Verlauf des Programmes so häufig verwendet, daß es sich lohnen würde, sie als Makro zu definieren. Eine Aufgabe für Sie! Von Zeile 870 bis 930 finden Sie eine kleine 8-Bit-Schleife, die unserer Variante 4 aus der ersten Folge entspricht. Der Job dieser Schleife ist das Beschreiben der ersten Zeile mit dem Zeichen 2. Wenn schließlich die Posi-

```
10   -.list 1,4                  ;in hypra-ass: .li 1,4
20   -.base $c000                ;in hypra-ass: .ba $c000
30   -;***********************************
40   -; verschachtelte schleifen  anwendung
50   -; allgemeiner bildschirmrahmen
60   -;***********************************
70   -;
80   -;lagerplatz fuer die zeichen: (hpyra-ass jeweils: .eq code1 = $1b usw.)
90   -.define code1     = $1b   ;zeichen O = 4f
100  -.define code2     = $1c   ;zeichen 7 = 77
110  -.define code3     = $1d   ;zeichen P = 50
120  -.define code4     = $1e   ;zeichen % = 74
130  -.define code5     = $1f   ;zeichen ' = 6a
140  -.define code6     = $20   ;zeichen L = 4c
150  -.define code7     = $21   ;zeichen / = 6f
160  -.define code8     = $22   ;zeichen : = 7a
170  -;vektoren fuer bildchirm- und farb-ram:
180  -.define screen    = $fb   ;bildschirmstart
190  -.define color     = $fd   ;farbramstart
200  -;zaehler:
210  -.define horiz     = $23   ;zaehler fuer horizontale
220  -.define vert      = $24   ;zaehler fuer vertikale
230  -;der farbcode wird nur im x-register gespeichert
240  -;
250  -;definition eines makro: aktuell
260  -;in hypra-ass stattdessen: .ma aktuell(screen,color)
270  -.macro aktuell(screen,color)
280  -            clc
290  -            tya
300  -            adc screen
310  -            sta screen
320  -            lda screen+1
330  -            adc #$00
340  -            sta screen+1
350  -            clc
360  -            tya
370  -            adc color
380  -            sta color
390  -            lda color+1
400  -            adc #$00
410  -            sta color+1
420  -            ldy #$00
430  -.macend                        ;in hypra-ass: .rt
440  -;
450  -;nun gehts los:
460  -;
470  -;-------------- initialisierung ------------
480  -;
490  -            ldx #$02       ;farbcode ins x-register
500  -            lda #$4f       ;alle zeichen in die speicher packen
510  -            sta code1
520  -            lda #$77
530  -            sta code2
540  -            lda #$50
550  -            sta code3
560  -            lda #$74
570  -            sta code4
580  -            lda #$6a
590  -            sta code5
600  -            lda #$4c
610  -            sta code6
620  -            lda #$6f
630  -            sta code7
640  -            lda #$7a
650  -            sta code8
660  -            lda #$00       ;lsb bildschirmstart
670  -            sta screen
680  -            lda #$04       ;msb bildschirmstart
690  -            sta screen+1
700  -            lda #$00       ;lsb farbramstart
710  -            sta color
720  -            lda #$d8       ;msb farbramstart
730  -            sta color+1
740  -            lda #39        ;stellenzahl pro zeile
750  -            sta horiz
760  -            lda #23        ;zeilenzahl-2
770  -            sta vert
780  -            ldy #$00       ;zaehler auf null
790  -;
800  -;------------ nun kommen die jobs: ------------
810  -;job 1: beschreiben der oberen zeile
820  -            lda code1      ;1.zeichen links oben
830  -            sta (screen),y ;einschreiben
840  -            txa            ;farbcode
850  -            sta (color),y  ;in farbramstelle schreiben
860  -            iny            ;zaehler erhoehen
870  -marke1      lda code2      ;2.zeichen einladen
880  -            sta (screen),y ;eine kleine schleife
890  -            txa
900  -            sta (color),y  ;farbram beschreiben
910  -            iny
920  -            cpy horiz      ;ist y schon > 38 ?
930  -            bcc marke1     ;wenn nicht, weitermachen
940  -            lda code3      ;letztes zeichen obere zeile
950  -            sta (screen),y ;nach rechts oben schreiben
960  -            txa
970  -            sta (color),y  ;wieder farbcode eintragen
980  -            iny
990  -;das war die erste zeile. nun kommen die seitenlinien
1000 -;job 2: beschreiben der seiten
1010 -;
1020 -marke2
1030 -.. aktuell(screen,color)  ;aufruf des makro
1035 -;in hypra-ass: ...aktuell(screen,color)
1040 -;damit werden die vektoren screen und color aktualisiert
1050 -            lda code4      ;4.zeichen fuer linken rand
1060 -            sta (screen),y ;eintragen
1070 -            txa            ;und den farbcode
1080 -            sta (color),y  ;ebenfalls
1090 -marke3      iny            ;zaehler schnell hochzaehlen
1100 -            cpy horiz      ;ist y schon > 38 ?
1110 -            bcc marke3     ;wenn nicht, weiterzaehlen
1120 -            lda code5      ;sonst 5.zeichen fuer rechten rand
1130 -            sta (screen),y ;eintragen
1140 -            txa            ;und farbcode
1150 -            sta (color),y  ;in farbram
1160 -            iny
1170 -            dec vert       ;mitzaehlen der zeilen
1180 -            bne marke2     ;naechste zeile, wenn noch nicht null
1190 -;ansonsten sind nun beide seitenlinien vorhanden
1200 -;job 3: beschreiben der letzten zeile
1210 -;dazu nochmal makroaufruf zum aktualisieren der vektoren
1220 -.. aktuell(screen,color)  ;dies ist der aufruf
1225 -;in hypra-ass wieder: ...aktuell(screen,color)
1230 -;nun kommt der job:
1240 -            lda code6      ;6.zeichen links unten
1250 -            sta (screen),y ;einschreiben
1260 -            txa            ;wieder den farbcode
1270 -            sta (color),y  ;ins farbram schreiben
1280 -            iny            ;und den zaehler erhoehen
1290 -marke4      lda code7      ;7.zeichen
1300 -            sta (screen),y ;eintragen. kleine schleife
1310 -            txa
1320 -            sta (color),y  ;farbram
1330 -            iny
1340 -            cpy horiz      ;schon > 38 ?
1350 -            bcc marke4     ;wenn nicht, weitermachen
1360 -            lda code8      ;sonst letztes zeichen
1370 -            sta (screen),y ;nach rechts unten schreiben
1380 -            txa            ;und auch den farbcode
1390 -            sta (color),y  ;ins farbram eintragen
1400 -;
1410 -;damit ist die arbeit getan
1420 -;--------------- ausgang einfach durch brk (monitor!) ---------
1430 -;
1440 -            brk
1450 -;
1460 -.symbols 1,4                ;in hypra-ass: .sy 1,4
```

Listing 18. Programm zum Zeichnen eines Rahmens um den Textbildschirm

tion 39 erreicht ist (also die 40. Spalte), wird die Schleife verlassen und das letzte Zeichen dieser Zeile eingetragen. Ein abschließendes INY richtet das Programm auf die erste Stelle der 2. Zeile.

Wir kommen damit zum Mittelteil, zum Beschreiben der beiden Seitenlinien. Weil wir ohnehin Y nicht durchgängig als Zähler verwenden können (es geht halt nur bis 255), verändern wir jetzt die beiden Vektoren SCREEN und COLOR, indem wir den aktuellen Stand des Y-Registers addieren. Danach können wir Y wieder auf Null setzen und HORIZ als

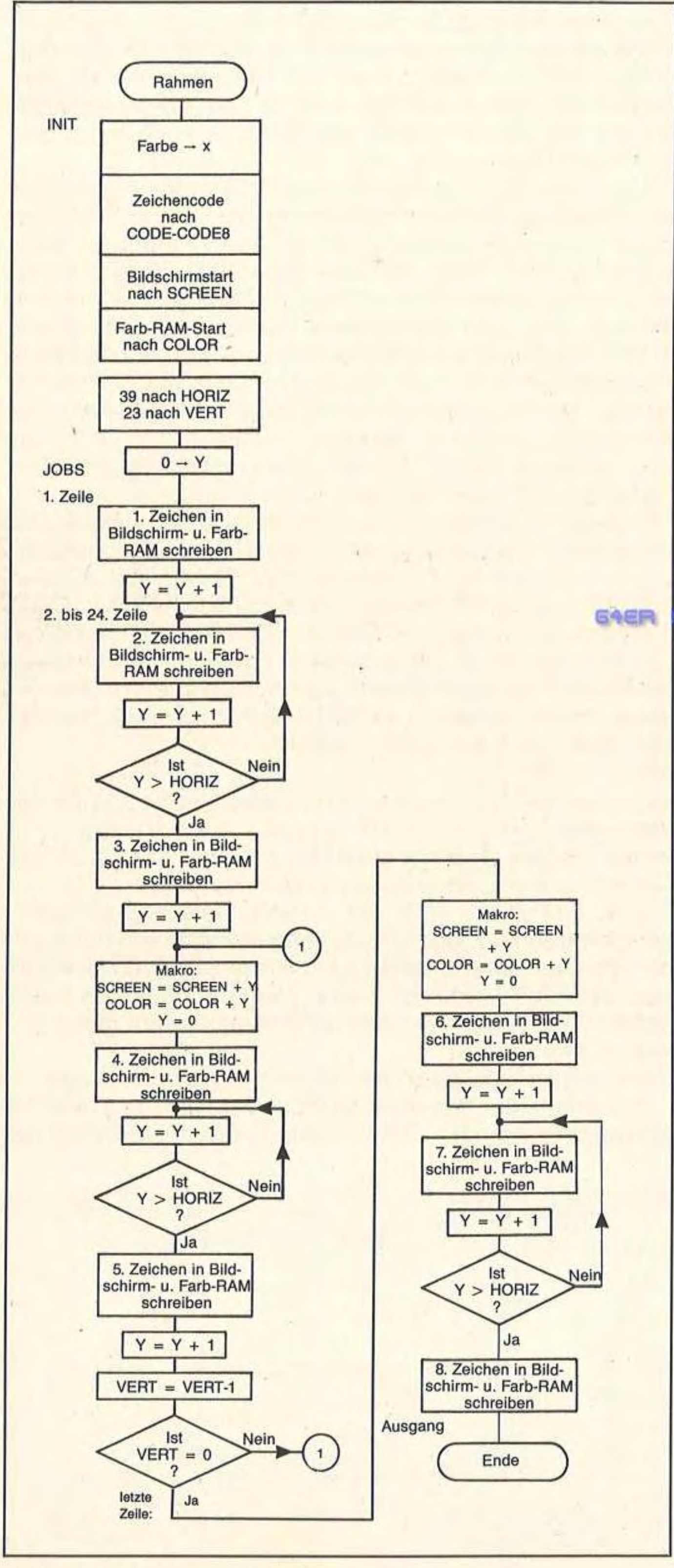


Bild 5. Flußdiagramm zum Programm RAHMEN

Grenzbedingung weiterverwenden. Diese Addition wird mehrfach gebraucht, deshalb ist sie zum Makro erklärt worden und ihre Definition als AKTUELL in den Zeilen 270 bis 430 angegeben. Jedesmal, wenn nun das letzte Zeichen einer Zeile eingetragen wurde, findet der Makroaufruf statt, deshalb ist in Zeile 1020 noch eine Sprungmarke angegeben. In Zeile 1030 liegt der Makroaufruf. Wenn dadurch die Vektoren aktualisiert wurden und das Y-Register wieder auf Null steht, schreiben wir in die erste Position der neuen Zeile das Zeichen 4. Es schließt sich wieder eine kleine 8-Bit-Schleife an (Zeilen 1090 bis 1110), in der lediglich Y hochgezählt wird. Anschließend tragen wir das letzte Zeichen der Zeile ein und ziehen eine 1 ab von der Gesamtzahl an Zeilen, die in VERT gespeichert wurde. Zeile 1180 prüft nun, ob VERT schon bis Null heruntergezählt ist. Falls nicht, fängt dieser Programmteil wieder von vorne an: Aktualisieren der Vektoren, Nullsetzen des Y-Registers, und so fort. Ansonsten aber sind die Seitenlinien fertig und wir können die untere Zeile bearbeiten. Aber halt! Einige erfahrene Assembler-Hasen werden schon mehrmals entrüstet das Haupt geschüttelt haben: Warum denn so unökonomisch? Nun, im Prinzip hätten wir anstelle der Zeilen 1090 bis 1110 auch einfach eine Addition vornehmen können, so daß alle Speicherstellen zwischen linkem und rechtem Rand einer Zeile einfach übersprungen werden, oder aber wir hätten diesen ganzen Mittelteil ganz anders angepackt, wenn da nicht noch eine interessante Möglichkeit bestehen würde: In die leere Y-Schleife könnte ja noch ein Job gepackt werden! So können Sie noch Texte ausgeben lassen oder ähnliches einschieben in die leere Schleife. Probieren Sie es mal aus!

Zur Bearbeitung der letzten Zeile rufen wir wieder das Makro zum Aktualisieren der beiden Vektoren auf. Der Rest ist genauso aufgebaut wie der Teil zum Beschreiben der ersten Zeile. Unser Rahmen ist fertig.

Sie haben aus der Programmbeschreibung sicher schon erkennen können, daß wir hier nur einen von vielen möglichen Wegen gegangen sind. Beispielsweise hätten wir auch mit Rückwärtsschleifen arbeiten können und damit die CPY-Befehle eingespart. Wir hätten die Programmteile für die erste und die letzte Zeile als ein Unterprogramm schreiben können, das mit verschiedenen Parametern aufgerufen wird, wir hätten... Beim Programmieren in Assembler hat man meistens viel mehr Variationsmöglichkeiten als in Basic. Zwar bleiben einige unveränderliche Skeletteile immer bestehen (ein normaler Textbildschirm fängt halt bei $400 an, zu beschreibende Register liegen immer an derselben Stelle, etc.), wie wir aber dieses Gerüst umhüllen, steht uns weitgehend frei. Wenn Sie unser Rahmenprogramm in der vorliegenden Form (am besten noch mit einem RTS anstelle des BRK am Ende) aufrufen, dann zeichnet es Ihnen einen roten Standardrahmen um den gesamten Textbildschirm. Benutzer des C 128 sollten das Programm nach $1400 (statt nach $C000) legen. Falls es nämlich von $C000 aus gestartet wird, bleibt die Rahmenfarbe unberücksichtigt, weil das Farb-RAM in einer anderen Bank liegt. Von $1400 aus kann man es dann durch

```
BANK 15: SYS DEC("1400")
```

starten und der Rahmen wird bunt. Soll Ihr Rahmen aus anderen Zeichen bestehen? Kein Problem: Dann ändern Sie die Initialisierung oder POKEn Sie direkt in das Programm die neuen Codes. Eine andere Rahmenfarbe wird gewünscht? Wieder können Sie durch POKE-Kommando oder Ändern der Initialisierung Ihre Farbe erzielen. Der Rahmen soll nur um die obere Bildschirmhälfte reichen? Dann verändern Sie einfach den Wert, der nach VERT geschrieben wird. Schwieriger wird es, wenn Sie auch HORIZ anders haben möchten. Nach dem Setzen des letzten Zeichens in eine Zeile müßte noch anstelle des INY eine

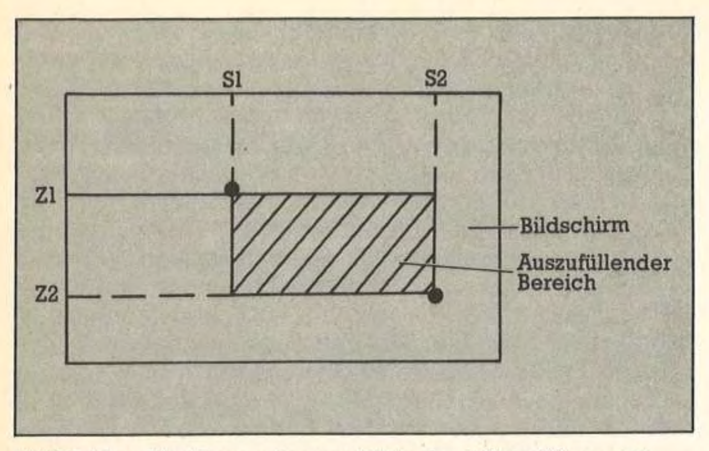


Bild 6. Zum Problem, wie man Teile des Bildschirms mit Zeichen füllt

Addition stehen, damit der Abstand zum ersten Zeichen der nächsten Zeile übersprungen wird. Vielleicht hilft Ihnen dazu die Antwort auf die nächste Frage.

Teilbereiche des Bildschirmes beschreiben

Ein weiteres Problem tritt in Leserfragen häufig auf: Wie kann man beliebige rechteckige Teile des Bildschirmes mit Zeichen ausfüllen? Sehen wir uns das also auch mal an: Bild 6 zeigt, was gemeint ist und welche Bezeichnungen wir im folgenden verwenden werden.

Die linke obere Ecke ist durch die Spalte S1 und die Zeile Z1, die rechte untere durch S2 und Z2 angegeben. Damit steht unser Rechteck fest. Drei Größen brauchen wir nun noch: Die Startadresse im Bildschirmspeicher und im Farb-RAM, die zu dem Punkt S1,Z1 gehört, die Breite und die Höhe des rechteckigen Bereiches. Das Bild 7 zeigt Ihnen die Arbeit, die wir im Speicher zu vollbringen haben:

Breite und Höhe zu berechnen, ist einfach:
Breite: SD = S2–S1
Höhe : ZD = Z2–Z1

Die Startadresse berechnet sich aus der Bildschirmadresse SC (1024), der Zeilenzahl Z1-1, die jeweils mal 40 zu nehmen ist und der Spaltenzahl S1:
Start : S = 40*(Z1-1) + S1

In Basic sähe unsere Problemlösung aus wie im Listing 19 gezeigt. C 64-Benutzer setzen in Zeile 80 statt des DEC ("3FF") einfach 1023 ein.

Natürlich funktioniert das einwandfrei, aber es dauert! In Assembler geht das Ausfüllen auch dann noch blitzartig, wenn wir als Rechteck den gesamten Bildschirm vorgeben. Das einzige, was Kopfzerbrechen bereiten kann, ist die Berechnung der Startadresse. Aber auch dazu gibt es einen schnellen Weg, wie Sie gleich noch sehen werden.

Bild 8 gibt Ihnen als Anhaltspunkt ein Flußdiagramm zum Assemblerprogramm TEILBEREICH.

Listing 20 schließlich enthält unser gleich zu besprechendes Programm »Teilbereich«.

Im ersten Teil des Listings sehen Sie wieder die Definitionen, die der Assembler beim Assemblieren statt der Merktexte einsetzt. Diesmal sind von Zeile 130 bis 180 Werte vorgegeben (purpurfarbene Zeichen A werden in das Rechteck geschrieben, das von Spalte 4/Zeile 3 bis Spalte 15/ Zeile 13 reicht), von 220 bis 270 Zeropage-Speicherstellen, die der Speicherung der Spaltendifferenz (SPDIFF), Zeilendifferenz (ZDIFF), der Spalte S1 (SPALTE) und eines Zwischenwertes der Rechnung (ZWSP) dienen. Außerdem sind zwei Vektoren vorgesehen: BILD (für die laufende Bildschirmspeicherposition) und COLOR (dasselbe für das Farb-RAM). Die Zeilen 310 und 320 enthalten die Festlegung der Basisadressen des Bildschirmspeichers und des Farb-RAMs (jeweils –1).

Der erste Teil des eigentlichen Programmes dient der Initialisierung und der Berechnung der Steuergrößen (im Basic-Programm waren das SD, ZD und S). Zunächst speichern wir Z1 als 16-Bit-Wert im Vektor BILD. Dazu muß das MSB von BILD (also BILD+1) noch auf Null gesetzt werden, denn Z1 ist ja nur ein 8-Bit-Wert. Der Wert S1 wird danach in SPALTE geschrieben (Zeilen 430 und 440). Nun fängt die Rechnerei an. Wir bilden die Spalten- und die Zeilendifferenzen jeweils durch einfache 8-Bit-Subtraktionen. Zur Erinnerung: Vor einer normalen Subtraktion muß immer das Carry-Bit gesetzt werden, denn es dient gewissermaßen als »Stelle, die geborgt werden kann«. Die Wirklichkeit ist etwas komplexer, aber das können Sie – falls es Sie interessiert – im Assemblerkurs oder in einschlägigen Lehrbüchern nachlesen. Nach vollendeter Subtraktion werden SPDIFF und ZDIFF jeweils noch durch den INC-Befehl um 1 erhöht. Das hat seinen Grund in den Schleifenabbruchbedingungen im 2. Teil unseres Programmes. Soweit war die Sache noch recht einfach. Nun kommt aber die Berechnung des Startpunktes im Bildschirm- und im Farb-RAM. Das heißt, wir haben den Ausdruck
40*(Z1-1)+S1
zu berechnen. Wir bedienen uns in der Hauptsache der Assembler-Befehle ASL und ROL dazu, die – weil wir sie recht selten erleben – hier nochmal kurz erläutert werden sollen. Die Bilder 9 und 10 illustrieren die Erklärungen.

ASL (arithmetic shift left = arithmetisches Linksverschieben) schiebt jedes Bit des angesprochenen Bytes um eine Position nach links. An die Stelle des Bit 0 tritt eine Null und das Bit 7 landet im Carry. Was das bedeuten kann, sehen wir uns an einer Zahl im Dezimalsystem mal an:
240 unsere Zahl
2400 nun ist jede Stelle einmal nach links verschoben

Sie sehen, daß das einer Multiplikation um den Faktor 10 entspricht. Im binären Zahlensystem, das unser Computer

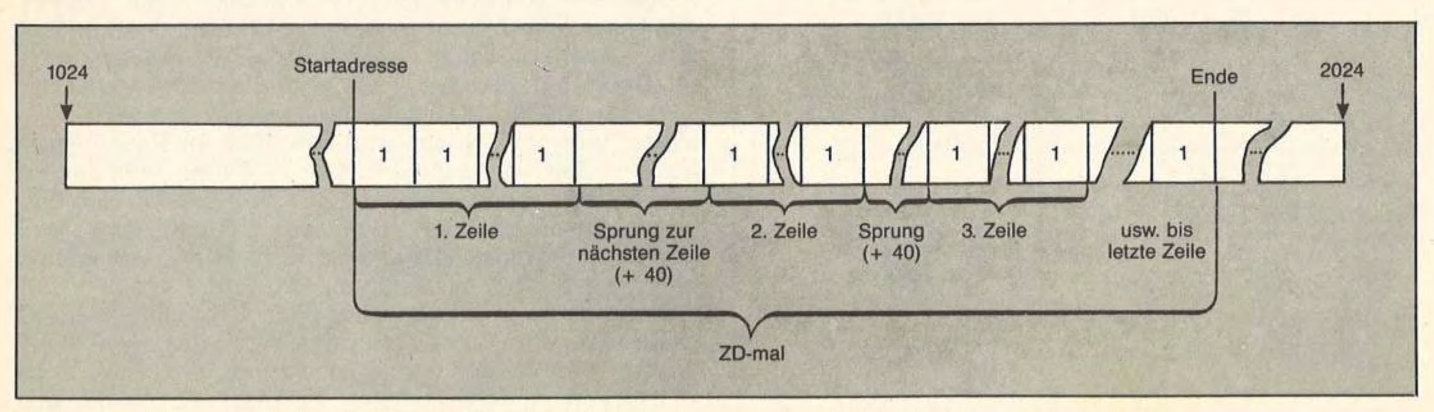


Bild 7. Das haben wir im Bildschirm-RAM zu tun. Hier ein Auszug, der eine Bildschirmspalte darstellt

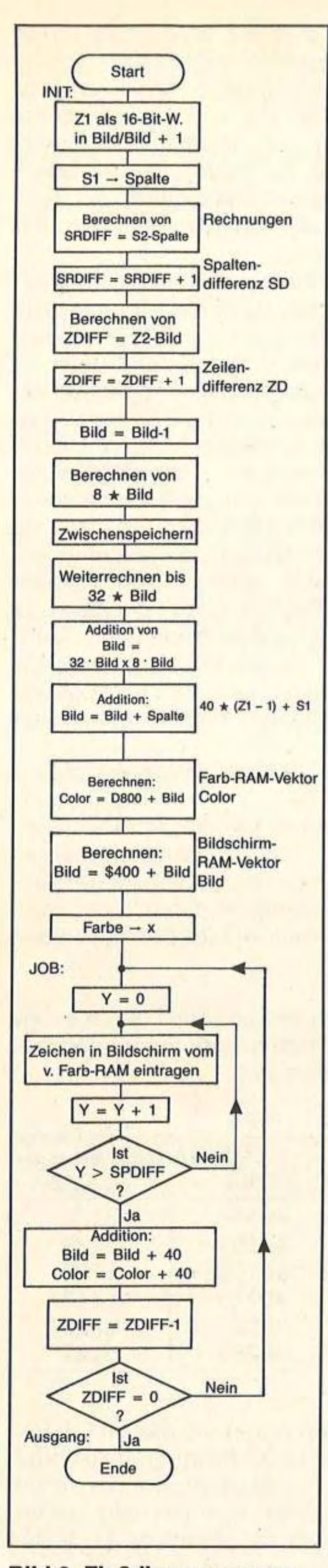


Bild 8. Flußdiagramm zum Programm »Teilbereich«

verwendet, ist die Zahlenbasis nicht mehr 10, sondern 2. Deshalb führt hier jede Linksverschiebung zu einer Multiplikation mit 2. Sehen wir uns das am konkreten Beispiel an: Der höchste Wert für Z1 in unserem Programm ist 25 (es gibt nur 25 Zeilen). Wenn wir von Z1-1 ausgehen, ist das 24:

0001 1000 unsere Zahl 24
1.ASL: 0011 0000 das ist 2*24
2.ASL: 0110 0000 das ist 4*24
3.ASL: 1100 0000 das ist 8*24

Vorsicht! Wenn wir nun nochmal die ASL-Operation anwenden, schieben wir das Bit 7 in das Carry-Bit. Das Ergebnis ist für eine 8-Bit-Zahl zu groß geworden, eine 16-Bit-Zahl ist notwendig. Wir müßten dann die 1 aus dem Carry-Bit als Bit 0 des MSB verwenden können. Das wird mit ROL (rotate left = nach links rotieren) möglich. Auch verschiebt der Befehl jedes Bit um eine Stelle nach links. Als neues Bit 0 aber tritt der Inhalt des Carry-Bit ein. Parallel dazu wandert das Bit 7 ins Carry-Bit. Der weitere Weg in unserem Beispiel muß nun also lauten:

4.ASL: 1 1000 0000 und dann
1.ROL: 0000 0001 1000 0000
MSB LSB

Das ist nun 16*24. Durch ASL wurde Bit 7 ins Carry geschoben (deshalb steht es oben links außen), durch ROL (bezogen auf die Speicherstelle, die als MSB eingesetzt wird) wandert es dann ins MSB als Bit 0. Die dazugehörende Befehlssequenz lautet (siehe beispielsweise Zeilen 650/660 des Listing 20):

ASL BILD
ROL BILD+1

Vermutlich werden Sie sich nun fragen, was das alles mit unserem Problem zu tun hat, 40*(Z1-1)+S1 zu berechnen. Sehr viel! Rechnen Sie es mal nach. Es gilt nämlich:

$40*(Z1-1) = 8*(Z1-1) + 32*(Z1-1)$

Sowohl 8 als auch 32 sind aber Potenzen von 2, das bedeutet, daß sie durch mehrmaliges Multiplizieren der 2 mit sich selbst herauskommen: 2↑3 = 8 und 2↑5 = 32. Es ist damit möglich, durch mehrmaliges Anwenden der ASL-Operation auf (Z1-1) beide Summanden zu erzeugen. Die Addition ergibt dann unseren gewünschten Wert 40*(Z1-1). Wir müssen nur noch S1 hinzurechnen. Wie wir vorhin noch sehen konnten, dürfen wir die drei ersten ASL-Operationen noch ohne Rücksicht auf Verluste durchführen. Danach allerdings muß jedem ASL das dazugehörige ROL folgen, denn das Ergebnis kann eine 16-Bit-Zahl sein.

```
10 REM *** PROGRAMM TEILBILDSCHIRM IN BASIC ***
20 INPUT"ECKE LINKS OBEN S1,Z13SPACE=";S1,Z1
30 INPUT"ECKE RECHTS UNTEN S2,Z2 =";S2,Z2
40 INPUT"WAS SOLL HINEIN ";W
50 REM ** STEUERWERTE BERECHNEN:
55 REM        S  = BILDSCHIRMSTARTADRESSE
60 REM        SD = SPALTENDIFFERENZ
70 REM        ZD = ZEILENDIFFERENZ
80 SC = DEC("3FF"):S = SC + 40*(Z1-1) + S1:SD = S2
  - S1:ZD = Z2 - Z1
90 REM *** SCHLEIFE ***
100 I=0:Z=0
110 :POKE S+I,W
120 :I=I+1:IF I>SD THEN S=S+40:I=0:Z=Z+1:IF Z>ZD T
HEN END
130 :GOTO 110
```

Listing 19. Ein Programm zum Füllen rechteckiger Bildausschnitte

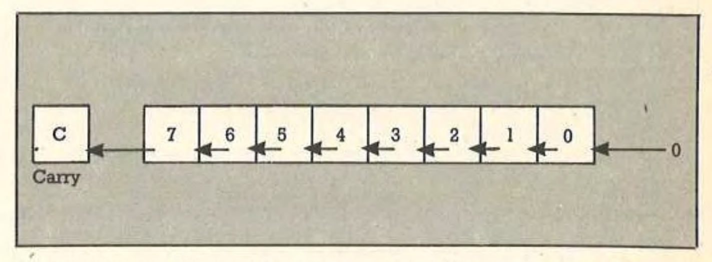


Bild 9. Die grafische Darstellung der Wirkung eines ASL-Befehls

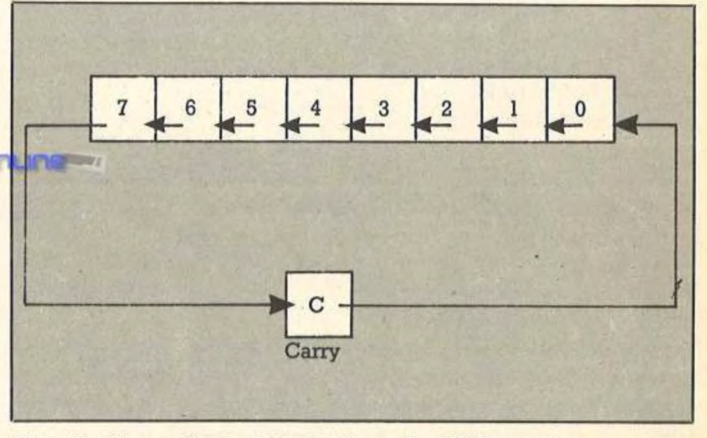


Bild 10. Die grafische Darstellung der Wirkung des ROL-Kommandos

In unserem Programm beginnen wir diese Berechnung in Zeile 580. Dort wird zunächst sicherheitshalber das Carry-Bit gelöscht für das erste ROL-Kommando. Durch DEC BILD erzeugen wir den Wert Z1-1 (nach BILD hatten wir ja Z1 geschrieben, BILD+1 wurde reserviert für das MSB und eine Null eingetragen). Dreimaliges ASL BILD erzeugt 8*(Z1-1). Sie erinnern sich: Erst vom vierten ASL an mußte beim größten erlaubten Z1-Wert auch ROL eingesetzt werden. In den Zeilen 640/650 packen wir diesen Summanden in den Zwischenspeicher. Danach sind noch zwei ASL mit anschließenden ROL-Kommandos nötig, um in BILD/BILD+1 den Wert 32*(Z1-1) zu erzeugen. Von Zeile 690 bis 740 addieren wir beide Summanden und erhalten so 40*(Z1-1). Normalerweise muß vor jeder Addition mit ADC (denn das heißt ja »add with carry«, also addiere mit dem Carry-Bit, sogar addiere auch das Carry-Bit) das Carry-Bit durch CLC freigemacht werden. Hier können wir uns das aber ersparen, weil wir wissen, daß auch der größte Z1-Wert bei der letzten ROL-Operation nur eine Null ins Carry-Bit schieben konnte. Bei der folgenden Addition

```
10   -;
20   -.list 1,4,7              ;in hypra-ass: .li 1,4,7
30   -;
40   -;*****************************************************************
50   -; schleifenprogrammierung beispiel teilbereiche des bildschirmes
60   -;                        beschreiben
70   -;*****************************************************************
80   -;
90   -.base $c000              ;in hypra-ass: .ba $c000
100  -;
110  -;---------- beispielwerte (durch pokes veraenderbar) ------------
120  -;
130  -.define farbe     = $04   ;in hypra-ass jeweils statt
140  -.define s1        = $04   ; .define immer:  .eq
150  -.define z1        = $03   ;spalte und zeile der ecke links oben
160  -.define s2        = $0f   ;spalte und
170  -.define z2        = $0c   ;zeile der ecke rechts unten
180  -.define wert      = $01   ;das ist der poke-code von a
190  -;
200  -;---------- speicherstellen ---------------------------------------
210  -;
220  -.define spdiff    = $fa   ;spaltendifferenz
230  -.define zdiff     = $fb   ;zeilendifferenz
240  -.define bild      = $fc   ;vektor aktuelle bildschirmposition
250  -.define spalte    = $fe   ;zwischenspeicher fuer s1
260  -.define zwsp      = $1b   ;zwischenspeicher fuer 8*z1
270  -.define color     = $1c   ;vektor farbramposition
280  -;
290  -;---------- adressen wichtiger ram-bereiche ----------------------
300  -;
310  -.define screen    = $3ff  ;bildschirmram - 1
320  -.define colram    = $d7ff ;farbram - 1
330  -;
340  -;****** jetzt faengt das programm an: ****************************
350  -;
360  -;initialisieren und berechnen der startadresse:
370  -;
380  -            lda #$00      ;z1 als 16-bit-wert speichern, msb
390  -            sta bild+1
400  -marke1      lda #z1       ;lsb. durch poke veraenderbar
410  -            sta bild
420  -;
430  -marke2      lda #s1       ;s1 speichern. durch poke veraenderbar
440  -            sta spalte
450  -;
460  -            sec           ;berechnung von spdiff
470  -marke3      lda #s2       ;durch poke zu aendern
480  -            sbc spalte
490  -            sta spdiff
500  -            inc spdiff    ;spaltenzaehler + 1
510  -;
520  -            sec           ;berechnung zdiff
530  -marke4      lda #z2       ;wieder durch poke zu aendern
540  -            sbc bild
550  -            sta zdiff
560  -            inc zdiff     ;zeilenzaehler + 1
570  -;
580  -            clc           ;carry freimachen fuer berechnung 40*z1
590  -            dec bild      ;z1=z1-1
600  -            asl bild      ;=z1*2
610  -            asl bild      ;=  *4
620  -            asl bild      ;=  *8 (erster summand)
630  -            lda bild
640  -            sta zwsp      ;zwischenspeichern
650  -            asl bild      ;=  *16
660  -            rol bild+1    ;zur sicherheit carry in msb rotieren
670  -            asl bild      ;=  *32 (zweiter summand)
680  -            rol bild+1    ;nochmal carry ins msb rotieren
690  -            lda zwsp      ;addition beider summanden (carry ist frei)
700  -            adc bild      ;lsb von 32*z1
710  -            sta bild      ;lsb von 40*z1
720  -            lda bild+1    ;msb von 32*z1
730  -            adc #$00      ;eventuell carry addieren
740  -            sta bild+1    ;im vektor bild steht nun 40*(z1-1)
750  -;
760  -            clc           ;addition von s1
770  -            lda bild      ;lsb
780  -            adc spalte    ;das ist ja s1
790  -            sta bild
800  -            lda bild+1    ;msb
810  -            adc #$00      ;eventuell carry addieren
820  -            sta bild+1    ;im vektor bild steht nun 40*(z1-1)+s1
830  -;
840  -            clc           ;farbramadresse berechnen
850  -            lda #<(colram) ;lsb
860  -            adc bild
870  -            sta color     ;vektor aktuelle farbramadresse lsb
880  -            lda #>(colram) ;msb
890  -            adc bild+1
900  -            sta color+1   ;vektor msb
910  -;
920  -            clc           ;aktuelle bildschirmadresse berechnen
930  -            lda #<(screen) ;lsb
940  -            adc bild
950  -            sta bild      ;vektor bildschirm lsb
960  -            lda #>(screen) ;msb
970  -            adc bild+1
980  -            sta bild+1    ;bildschirmvektor msb
990  -;
1000 -marke5      ldx #farbe    ;durch poke veraenderbar
1010 -;
1020 -;******* hier beginnen die verschachtelten schleifen *******
1030 -label1      ldy #$00      ;spaltenzaehler zuruecksetzen
1040 -label2      lda #wert     ;durch poke veraenderbar
1050 -            sta (bild),y  ;auf bildschirm schreiben
1060 -            txa           ;farbwert umladen
1070 -            sta (color),y ;und in farbram schreiben
1080 -            iny
1090 -            cpy spdiff    ;zeile fertig ?
1100 -            bcc label2    ;noch nicht, dann weitermachen
1110 -;ansonsten werden zu den vektoren bild und color 40 addiert:
1120 -            clc           ;bild aktualisieren
1130 -            lda bild      ;lsb
1140 -            adc #$28      ;das ist dezimal 40
1150 -            sta bild
1160 -            lda bild+1    ;msb
1170 -            adc #$00      ;carry addieren
1180 -            sta bild+1    ;vektor bild ist aktualisiert
1190 -;
1200 -            clc           ;color aktualisieren
1210 -            lda color     ;lsb
1220 -            adc #$28      ;wieder 40 dazuzaehlen
1230 -            sta color
1240 -            lda color+1   ;msb
1250 -            adc #$00      ;addieren des carry
1260 -            sta color+1   ;vektor color ist aktualisiert
1270 -;
1280 -            dec zdiff     ;zeilenzaehler herunterzaehlen
1290 -            bne label1    ;wenn noch > 0, naechste zeile schreiben
1300 -;
1310 -;im anderen fall ist die aufgabe erledigt:
1320 -            rts           ;zurueck zum aufrufenden programm oder basic
1330 -;
1340 -.symbols u,1,4,7         ;in hypra-ass: .sy 1,4,7
```

Listing 20. Das Assembler-Programm »Teilbereich«

von S1 (Zeilen 760 bis 820) entsteht schließlich unser gewünschter Startwert – aber noch ohne Berücksichtigung der Anfangsadressen von Bildschirm- und Farb-RAM. Das kommt nun: Von Zeile 840 bis 900 entsteht im Vektor COLOR/COLOR+1 der Zeiger auf die erste Farb-RAM-Stelle, von Zeile 920 bis 980 im Vektor BILD/BILD+1 schließlich der auf die erste Bildschirmspeicherstelle. Zum Abschluß dieser Programmphase schieben wir noch den Farbcode ins X-Register.

Nun kommt die Doppelschleife, die den Bildschirmausschnitt beschreibt. Das Y-Register dient als Zähler jeweils einer Zeile. Der Zeichencode gelangt – wie der Farbcode – nach dem gleichen Rezept in die notwendigen Speicherstellen, das wir auch schon in den vorangegangenen Programmbeispielen gezeigt haben (das war die Sequenz, die wir dort auch als Makro hätten schreiben können: Hier in den Zeilen 1050 bis 1080). Jeweils am Ende der kleinen Schleife (in 1090 und 1100) wird der aktuelle Y-Zählerstand verglichen mit der Spaltendifferenz (SPDIFF) und – falls die Zeile noch nicht fertig ist – der nächste Durchlauf eingeleitet. Andernfalls addieren wir sowohl zum Vektor BILD/BILD+1 als auch zu COLOR/COLOR+1 den Wert 40, rücken also eine Zeile runter. In 1280 dient ein DEC ZDIFF dazu, die Anzahl bearbeiteter Zeilen abzustreichen, bis dabei eine 0 auftritt. Ist das noch nicht der Fall, wird zurückverzweigt nach LABEL1, wo wir den Y-Zähler neu initialisieren.

Der letzte Teil unseres Programmes besteht lediglich wieder aus einem lapidaren RTS.

Wenn Sie unser Programm nach dem Assemblieren einfach mittels SYS 49152 starten, sind automatisch die Beispielwerte eingesetzt. C 128-Benutzern sei wieder geraten, das Programm nach $1400 zu legen, damit auch der Farbspeicher belegt werden kann. Hier erfolgt der Start dann durch

```
BANK 15: SYS DEC("1400").
```

Sehr flexibel wird »Teilbildschirm« aber erst durch einige POKE-Kommandos ins Programm hinein. Folgende Speicherstellen werden dann angesteuert:

Parameter	Ort im Programm	POKE-Adressen C 64	C 128
Z1	MARKE1 + 1	49157	5125
S1	MARKE2 + 1	49161	5129
S2	MARKE3 + 1	49166	5134
Z2	MARKE4 + 1	49175	5143
Farbe	MARKE5 + 1	49255	5223
Wert	LABEL2 + 1	49259	5227

Vorausgesetzt wird bei diesen Angaben, daß im C 64 ab $C000 und im C 128 ab $01400 unser Programm zu finden ist. Listing 21 zeigt Ihnen, wie ein Basic-Programm aussehen kann, das alle POKE-Befehle ausführt und unsere Routine ansteuert. Falls geplant ist, Benutzer an dieses Programm zu lassen, die es nicht kennen, sollten Sie auch noch eine Überprüfung der Eingabewerte einbauen, andernfalls könnte beispielsweise Unsinn herauskommen, wenn S2 kleiner als S1 angegeben wird oder mal eine Zeilennummer, die größer als 25 ist. Die erlaubten Werte sind jeweils Spaltenwerte zwischen 1 und 40, Zeilen zwischen 1 und 25.

Wie man Schleifen in Assembler anwenden kann und auf wie verschiedene Arten das möglich ist, haben Ihnen diese vier Beispiele sicher zeigen können. Aber mit Schleifen ist noch viel mehr machbar.

Nun ist es endlich soweit: Wie versprochen, lernen Sie jetzt die Blockverschieberoutine kennen, aber auch ihre

```
10 REM ******** AUFRUFPROGRAMM FUER TEILBILDSCHIRM
**********
20 REM IN REM-BEMERKUNGEN JEWEILS WERTE FUER DEN C
128
30 REM VORAUSSETZUNG, DASS BEIM C64 AB $C000
40 REM                    BEIM C128 AB $1400 DAS
PROGRAMM LIEGT.
50 REM
60 REM ------- EINGABE DER PARAMETER ---------
70 PRINT CHR$(147):INPUT"LINKE OBERE ECKE Z1,S1";Z
1,S1
80 INPUT"RECHTE UNTERE ECKE Z2,S2";Z2,S2
90 INPUT"FARBE,WERT";F,W
100 REM ------- EINPOKEN DER WERTE ------------
110 POKE 49157,Z1:REM POKE 5125,Z1
120 POKE 49161,S1:REM POKE 5129,S1
130 POKE 49166,S2:REM POKE 5134,S2
140 POKE 49175,Z2:REM POKE 5143,Z2
150 POKE 49255,F:REM POKE 5223,F
160 POKE 49259,W:REM POKE 5227,W
170 REM ------ AUFRUF DER ROUTINE ------------
180 SYS49152
190 END
```

Listing 21. Ein Basic-Programm, das alle POKE-Befehle ausführt und unsere Routine ansteuert

Schwächen und einen Weg, Speicherbereiche fehlerfrei zu verschieben.

Eng mit dem Verschieben von Bereichen ist das andere Programm verwandt, das wir entwickeln. SWAP nennen wir es, und es soll Speicherbereiche miteinander vertauschen. Wie immer, so sind auch diesmal die Programme sowohl auf dem C 64 als auch dem C 128 einsetzbar.

Speicherblöcke verschieben

Häufiges Thema in Leseranfragen ist das Verschieben von Speicherbereichen. Das ist durchaus zu verstehen, denn mit einem Programminstrument, das beliebige Inhalte beliebig großer Speicherbereiche verschieben kann, läßt sich allerhand anstellen. So könnte man ein Basic-Programm vorübergehend beispielsweise nach $C000 legen, in der Zwischenzeit ein anderes laden und bearbeiten und dann das erste wieder herunterholen in den Basic-Speicher. Oder es wäre möglich, einen Hilfsbildschirm zu erstellen, diesen irgendwo im Speicher an einen sicheren Ort zu verlagern und ihn dann auf Tastendruck wieder hervorzuholen. Oder man könnte sich verschiedene Teile von Bildern erstellen, im Speicher ablegen und bei Bedarf in die aktuelle Bitmap blenden. Oder... Ihnen fallen bestimmt noch viele Anwendungen für ein solches Programminstrument ein.

Diejenigen unter Ihnen, vor denen nun ein C 64 steht, haben Glück: Im Betriebssystem des C 64 ist nämlich eine komplette und vor allem leicht ansteuerbare Blockverschieberoutine enthalten. C 128-Benutzer finden solch eine Routine zwar auch in ihrem Speicher vor (nämlich ab $F4EA8), die ist aber leider nicht zu verwenden, weil sie nicht einfach mit einem RTS endet, sondern noch allerlei unerwünschte Zeigeränderungen anstellt. Allerdings kann der C 128-Besitzer auch mit erheblichen Effekt auf den T-Befehl des eingebauten Monitors zugreifen. Auch von Basic aus ist das mit Hilfe des »programmierten Direktmodus« möglich. Wer sich stärker dafür interessiert, der sollte mal in folgendem Buch das Kapitel dazu nachlesen: Ponnath, »Grafikprogrammierung C 128«, Markt und Technik Verlag, MT857. Eine andere Möglichkeit für den C 128-Benutzer ist unser später noch vorzustellendes Programm BLOCK.

Sehen wir uns nun zunächst die im C 64-Interpreter enthaltene Blockverschieberoutine BLTUC an:

Name	BLTUC
Zweck	Verschieben von Speicherinhalten im Speicher
Adresse	$A3BF, dez. 41919
Vorbereitungen	Quelle Startadresse nach $5F/60
Endadresse+1	nach $5A/5B
Ziel	Endadresse+1 nach $58/59
Speicherstellen	$58 bis 5B, $5F, $60, $22
Register	Akku, X- und Y-Register
Stapelbedarf	keiner

Das scheint also der Weg zur Benutzung dieser Routine zu sein: Man schreibt ein Basic-Programm, das die leidige Umrechnung der drei Adressen (Quellenstart, Quellenende+1 und Zielende+1) übernimmt und die errechneten LSB und MSB in die erforderlichen Abholspeicherstellen packt. Danach braucht man nur noch mittels eines SYS 41919 die BLTUC-Routine zu starten. Sollten Sie es mal probieren wollen, dann werden Sie einen Absturz des Programmes erleben. So geht es nicht, und zwar deshalb, weil der Basic-Interpreter die Speicherstellen $5F und $60 nach dem Belegen mit der Quellenstartadresse mit seinen Merkwerten überschreibt. Glücklicherweise enthält aber die Seite 3 eine Möglichkeit, Werte abholbereit für die Register so aufzubewahren, daß sie nach einem SYS-Befehl im Akku, dem X- und dem Y-Register zu finden sind. Die Zuordnung ist dann so:

Name	Adresse $	Adresse dez.	Register
SAREG	30C	780	Akku
SXREG	30D	781	X-Register
SYREG	30E	782	Y-Register
SPREG	30F	783	Stapelzeiger

Wir schreiben nun die Quellenstartadresse statt nach $5F/60 zunächst nach 780 und 781. Der anschließende SYS-Befehl ruft zuerst ein kleines Maschinenprogramm auf, das die Werte in die richtigen Speicherzellen schreibt und dann BLTUC anspringt:

```
STA $5F
STX $60
JMP $A3BF
```

Beiliegend finden Sie ein kleines Basic-Programm, das alle diese Aufgaben übernimmt: »BLTUC BAS« (Listing 22)

BLTUC BAS zeigt die Funktion von BLTUC anhand des Bildschirmspeichers. In den Zeilen 40 und 50 wird in die erste Bildschirmzeile – ab Position 1025 – eine fortlaufende Reihe von verschiedenen Zeichen geschrieben, die wir im folgenden verschieben werden. Damit diese Zeichen sichtbar werden, müssen einige ältere Versionen des C 64 auch den Farbspeicher beschreiben. Das geschieht in der Zeile 40. Die Zeilen 54 und 56 erzeugen das kleine Maschinenprogramm, das die Belegung der Abrufzellen $5F, $60 und den Sprung in die BLTUC-Routine ausführt. Sie lesen den Dezimalcode des Maschinenprogrammes aus der DATA-Zeile in den Speicher ab 49152. Nun bereiten wir die erste Verschiebung vor: Hier soll einfach der ganze Bereich von 1025 bis 1063 um eine Zeile weiter geschoben werden, also nun bei 1065 beginnen. Damit alles nicht ganz so schnell geht, sind noch kleine Warteschleifen ins Programm eingebaut. In den Zeilen 90 bis 110 trennen wir die in 70 und 80 benannten Start- und Endadressen auf in die MSB- und LSB-Werte und schreiben sie in die erforderlichen Speicherstellen 88 bis 91, beziehungsweise 780 und 781 ein. Zeile 120 vollführt nun mittels des SYS-Aufrufes die

Verschiebung, was Sie auf dem Bildschirm erkennen können.

Auf Tastendruck gelangen Sie in den zweiten, den kritischen Teil des Programmes. Hier werden wir einen Fehler der BLTUC-Routine finden. Wir verschieben in diesem Teil den Inhalt des Speicherbereiches 1025 bis 1063 um eine Position abwärts, also in den Bereich 1024 bis 1062. Woran liegt es, daß hier plötzlich eine Fehlfunktion auftritt? Sehen wir uns dazu die BLTUC-Routine genauer an. Als Programm BLTUC (Listing 23) finden Sie nachstehend ein Disassemblerlisting der BLTUC-Routine, wie sie im C 64-Speicher ab $A3BF zu finden ist.

Die Anatomie der BLTUC-Routine

Das ganze Programm besteht aus zwei Teilen. Im ersten Teil werden Berechnungen angestellt über die Länge des zu transportierenden Bereiches und zwei Transportzeiger eingerichtet. Im zweiten Teil findet dann die eigentliche Verschiebung statt. Die erste 16-Bit-Subtraktion (Quelle bis Ende+1 minus Quelle-Start) legt das MSB der Länge ins X-Register (das enthält dann die Anzahl der zu transportierenden Pages) und das LSB ins Y-Register und in die Speicherstelle $22 (dort liegt dann die restliche Länge, die weniger als eine ganze Page beträgt). Der BEQ-Befehl stellt fest, ob überhaupt ein solcher Rest vorhanden ist und verzweigt ansonsten direkt in den Transportteil. Zwei weitere Subtraktionen (Quelle-Ende+1 minus Länge des Restes und Ziel-Ende+1 minus Länge des Restes) richten die Zeiger $5A/5B und $58/59 auf die Adressen der nächstniedrigeren ganzen Page. Der Rest befindet sich noch im Y-Register. Das X-Register dient als Page-Zähler. Der BCC-Befehl bei $A3E6 führt immer zum Sprung nach $A3EC, weil an dieser Stelle das Carry-Bit immer frei ist.

Danach beginnt der Transportteil. Er besteht im wesentlichen aus zwei ineinander verschachtelten Schleifen, von denen die innere Schleife Byte für Byte aus dem Quell- in den Zielbereich kopiert (dabei beginnt sie mit dem Rest), die äußere zunächst ebenfalls ein Byte überträgt und dann die MSB-Werte der beiden Zeiger ($59 und $5B) herunterzählt. Dabei wird auch jedesmal der Pagezähler (X-Register) um 1 reduziert.

Kopieren von oben und von unten

Wir stellen also fest, daß ein Bereich durch BLTUC immer von der höheren zur niedrigeren Adresse hin durchgearbeitet wird. Sowohl der Index Y als auch der Page-Zähler X werden heruntergezählt. Welche Folgen das hat, werden wir nun bei einer genauen Betrachtung aller möglichen Verschiebungsfälle schnell erkennen. Insgesamt acht sind zu unterscheiden:

1. Quell- und Zielbereich überschneiden sich nicht. Der Zielbereich liegt oberhalb des Quellbereiches. Das Kopieren erfolgt von unten (also von der niedrigsten Adresse an aufwärts. Die Register werden hochgezählt). Das nennen wir den Fall 1.
2. Gleiche Bedingungen wie in Fall 1. Aber das Kopieren geschieht von oben (also von der höchsten Adresse an abwärts. Die Register zählen wir hier herunter). Dies ist Fall 2.
3. Wieder liegt keine Überschneidung vor. Der Zielbereich liegt nun aber unterhalb des Quellbereiches. Das Kopieren erfolgt von unten. Fall 3 liegt vor.
4. Die Bedingungen sind mit Fall 3 identisch, aber es wird wieder abwärts kopiert. Das ist Fall 4.
5. Quell- und Zielbereich überschneiden sich. Ansonsten sind die Verhältnisse wie bei Fall 1. Das wäre dann Fall 5.
6. Fall 6 tritt ein, wenn gleiche Bedingungen wie in Fall 2 vorliegen. Einziger Unterschied ist auch hier die Überschneidung von Quell- und Zielbereich.
7. Fall 7 entspricht dem Fall 3 mit Überlappung der Bereiche.
8. Das ist wieder der Fall 4 mit der Überschneidung von Quell- und Zielbereich.

Die Fälle 1 bis 4 bereiten keine Probleme. Hier bleibt es uns überlassen, wie wir eigene Verschiebungsprogramme organisieren wollen. Der BLTUC-Routinenanwendung entsprechen die Fälle 2 und 4. Sehen wir uns nun Fall 5 an (siehe dazu Bild 11).

```
10 REM ***** BLTUC - TESTPROGRAMM *****     <179>
20 PRINT CHR$(147)                           <049>
30 POKE 53280,0:POKE 53281,5:POKE 646,1      <120>
35 REM +++ FARBRAM U. SCHIRM BELEGEN +++     <053>
40 FOR I = 0 TO 79:POKE 55296+I,1:NEXT I     <221>
50 FOR I = 1 TO 39:POKE 1024+I,I:NEXT I      <188>
52 REM +++ ML-PRG.PARAMETERUEBERGABE +++     <073>
54 FOR I=49152 TO 49158:READ A:POKE I,A:NE
   XT I                                      <219>
56 DATA 133,95,134,96,76,191,163             <002>
60 REM +++ AUFWAERTS VERSCHIEBEN +++         <188>
62 FOR I=0 TO 300:NEXT I                     <161>
64 PRINT CHR$(17)CHR$(17)"WIR VERSCHIEBEN
   JETZT!"                                   <196>
66 FOR I=0 TO 300:NEXT I                     <165>
70 QS=1025:QE=1064:REM QUELLE START UND QU
   ELLE ENDE+1                               <062>
80 ZE=1065:REM ZIEL ENDE+1                   <092>
90 A=INT(QS/256):POKE 781,A:POKE 780,QS-25
   6*A                                       <015>
100 A=INT(QE/256):POKE 91,A:POKE 90,QE-256
    *A                                       <099>
110 A=INT(ZE/256):POKE 89,A:POKE 88,ZE-256
    *A                                       <034>
120 SYS 49152                                <178>
130 PRINT CHR$(17)CHR$(17)"DAS WAR UM 1 AU
    FWAERTS":PRINT"BITTE TASTE DRUECKEN"     <167>
140 GET A$:IF A$=""THEN 140                  <014>
150 REM +++ ABWAERTS VERSCHIEBEN +++         <121>
152 FOR I=0 TO 300:NEXT I                    <251>
154 PRINT CHR$(17)CHR$(17)"JETZT VERSCHIEB
    EN WIR ABWAERTS!"                        <200>
156 FOR I=0 TO 300:NEXT I                    <255>
160 QS=1025:QE=1064:REM QUELLE START UND Q
    UELLE ENDE+1                             <152>
170 ZE=1063:REM ZIEL ENDE+1                  <180>
180 A=INT(QS/256):POKE 781,A:POKE 780,QS-2
    56*A                                     <105>
190 A=INT(QE/256):POKE 91,A:POKE 90,QE-256
    *A                                       <189>
200 A=INT(ZE/256):POKE 89,A:POKE 88,ZE-256
    *A                                       <124>
210 SYS 49152                                <012>
220 PRINT CHR$(17)CHR$(17)"DA SEHEN SIE DA
    S PROBLEM":PRINT"DER BLTUC-ROUTINE"      <177>
```

© 64'er

Listing 22. »BLTUC BAS« – Ein kleines Basic-Programm zum Testen der BLTUC-Routine

```
., a3bf 38        sec          ., a3dc a5 58     lda $58
., a3c0 a5 5a     lda $5a      ., a3de e5 22     sbc $22
., a3c2 e5 5f     sbc $5f      ., a3e0 85 58     sta $58
., a3c4 85 22     sta $22      ., a3e2 b0 08     bcs $a3ec
., a3c6 a8        tay          ., a3e4 c6 59     dec $59
., a3c7 a5 5b     lda $5b      ., a3e6 90 04     bcc $a3ec
., a3c9 e5 60     sbc $60      ., a3e8 b1 5a     lda ($5a),y
., a3cb aa        tax          ., a3ea 91 58     sta ($58),y
., a3cc e8        inx          ., a3ec 88        dey
., a3cd 98        tya          ., a3ed d0 f9     bne $a3e8
., a3ce f0 23     beq $a3f3    ., a3ef b1 5a     lda ($5a),y
., a3d0 a5 5a     lda $5a      ., a3f1 91 58     sta ($58),y
., a3d2 38        sec          ., a3f3 c6 5b     dec $5b
., a3d3 e5 22     sbc $22      ., a3f5 c6 59     dec $59
., a3d5 85 5a     sta $5a      ., a3f7 ca        dex
., a3d7 b0 03     bcs $a3dc    ., a3f8 d0 f2     bne $a3ec
., a3d9 c6 5b     dec $5b      ., a3fa 60        rts
., a3db 38        sec
```

Listing 23. »BLTUC« – So steht die BLTUC-Routine im C64-Speicher (Disassembler-Listing)

In Bild 11a ist die Ausgangslage abgebildet, wobei wegen der besseren Übersicht Quell- und Zielbereich untereinander gezeichnet sind. Natürlich handelt es sich bei den untereinanderliegenden Kästchen immer um ein und dieselbe Speicherstelle. In Bild 11b wird das erste Byte des Quellbereiches in die erste Speicherstelle des Zielbereiches kopiert. Das ist aber gleichzeitig das zweite Byte des Quellbereiches. Was nun geschieht, zeigen die Teilbilder 11c und schließlich 11d: Der gesamte Zielbereich füllt sich mit dem Inhalt der ersten Quellbereichs-Speicherstelle. Hätten Sie das gedacht?

Bild 12 verdeutlicht uns den Fall 6.

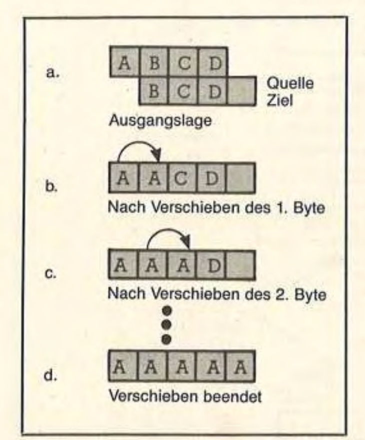


Bild 11. Der Fall 5

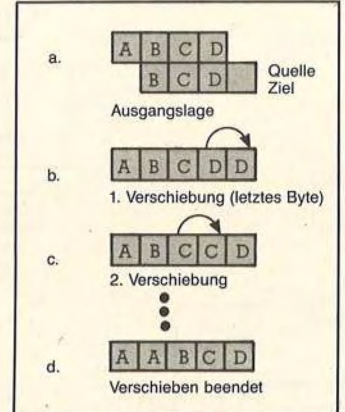


Bild 12. Der Fall 6

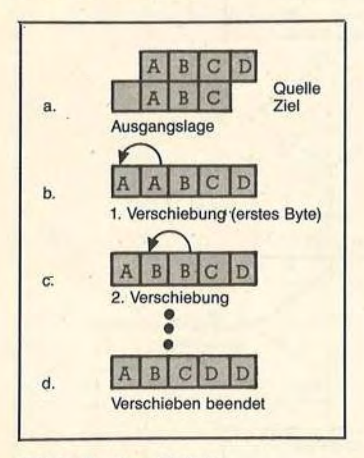


Bild 13. Der Fall 7

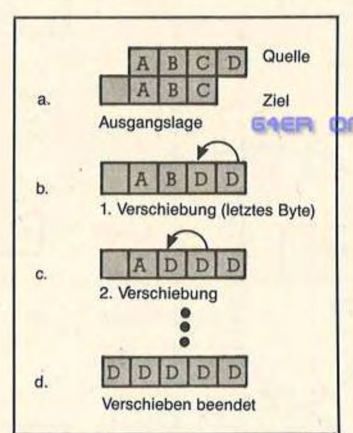


Bild 14. Der Fall 8

Es ist nach dem gleichen Schema wie Bild 11 aufgebaut. Sie sehen, daß nun aber von oben herunter gearbeitet wird. Die erste Verschiebung packt das letzte Byte des Quellbereiches in die letzte Speicherstelle des Zielbereiches (Teilbild b). An den folgenden Teilbildern c und d ist deutlich, daß diese Methode fehlerfrei funktioniert. Nach diesem Schema arbeitet die BLTUC-Routine, weshalb wir beim Aufwärtsverschieben von Speicherinhalten auch bei Überlappungen keine Störungen erwarten brauchen.

Wenden wir uns nun dem Fall 7 zu. Bild 13 soll bei dieser Betrachtung wieder helfen:

In Fall 7 liegt der Zielbereich unterhalb des Quellbereiches und es wird von unten gearbeitet, also die Register aufwärts gezählt. Aus Bild 14 ist – gleiches Schema wie bisher – zu entnehmen, daß keine Probleme auftreten. Zu guter Letzt hilft uns das Bild 14 zum Verstehen des Falls 8:

Im Teilbild 14b erkennen Sie das Problem: Sobald das letzte Byte des Quellbereiches in die letzte Speicherstelle des Zielbereiches verschoben ist, haben wir das vorletzte Byte des Quellbereiches damit überschrieben, denn das ist ja gleichzeitig die letzte Speicherstelle des Zielbereiches. Jede weitere Verschiebung kopiert nur wieder diesen gleichen Inhalt, was Ihnen die Teilbilder c und d zeigen. Genau das macht die BLTUC-Routine, wie Sie im zweiten Teil des Programmes BLTUC BAS feststellen konnten. Man darf also diese Interpreter-Routine nicht anwenden, wenn der Zielbereich unterhalb des Quellbereiches liegt und beide sich überschneiden!

Wenn es daher unsicher ist, ob sich Quell- und Zielbereich überlappen oder wenn man davon ausgehen kann, daß das sicher der Fall sein wird, dann verfahre man beim Erstellen eigener Verschiebe-Routinen nach folgender Regel:

- Abwärts kopieren beim Aufwärtsverschieben
- Aufwärts kopieren beim Abwärtsverschieben

Fehlerfreies Verschieben mit »BLOCK«

Wie sollte ein Verschiebeprogramm aussehen, das allen Eventualitäten gerecht wird? Ganz einfach: es müßte zunächst prüfen, ob eine Überlappung von Quell- und Zielbereich vorliegt und je nach Ergebnis dann zum entspre-

```
10   -;
20   -;
30   -.BASE $1300              ;IN HYPRA-ASS: .BA $C000
40   -;**************************************
50   -; BLOCKVERSCHIEBEROUTINE OHNE FEHLER
60   -;**************************************
70   -.DEFINE MVELEN    = $FA   ;IN HYPRA-ASS WIRD STATT DER
80   -.DEFINE MVDEST    = $FC   ; .DEFINE BEFEHLE JEWEILS DER BEFEHL
90   -.DEFINE MVSRCE    = $FE   ; .EQ = $... VERWENDET
100  -;
110  -;IN MVELEN WIRD DIE LAENGE DER ZU VERSCHIEBENDEN BEREICHES ANGEGEBEN
120  -;IN MVDEST DIE STARTADRESSE DES ZIELBEREICHES UND IN
130  -;MVSRCE DIE STARTADRESSE DES QUELLBEREICHES.
140  -;
150  -;------ PROGRAMM ----------------------------------------------
160  -;ALS ERSTES WIRD BESTIMMT, OB DER ZIELBEREICH OBERHALB DES
170  -; QUELLBEREICHES LIEGT UND OB SICH DIE BEIDEN BEREICHE UEBER-
180  -; LAPPEN. EINE UEBERLAPPUNG LIEGT DANN VOR, WENN DIE DIFFERENZ
190  -; VON ZIELADRESSE MINUS QUELLADRESSE KLEINER ALS DIE ANZAHL DER
200  -; ZU VERSCHIEBENDEN BYTES IST.
210  -;
220  -START      LDA MVDEST     ;BERECHNUNG ZIEL MINUS QUELLE
230  -          SEC
240  -          SBC MVSRCE
250  -          TAX
260  -          LDA MVDEST+1
270  -          SBC MVSRCE+1
280  -          TAY
290  -          TXA            ;VERGLEICH MIT LAENGE DES VERSCHIEBEBEREICHES
300  -          CMP MVELEN
310  -          TYA
320  -          SBC MVELEN+1
330  -          BCS DOLEFT     ;VERZWEIGEN, WENN KEINE UEBERLAPPUNG
340  -          JSR MVERHT     ;SONST ZUM UP FUER UEBERLAPPUNG
350  -          JMP EXIT
360  -DOLEFT     JSR MVELFT     ;ZUM UP OHNE UEBERLAPPUNG
370  -EXIT       RTS
380  -;
390  -;**** UP ZUM VERSCHIEBEN OHNE UEBERLAPPUNG: MVELFT ****
400  -;
410  -MVELFT     LDY #0         ;INDEX AUF NULL
420  -          LDX MVELEN+1   ;ANZAHL PAGES IN X
430  -          BEQ MLPART     ;FALLS KEINE GANZEN PAGES DANN REST
440  -MLPAGE     LDA (MVSRCE),Y ;EIN BYTE VERSCHIEBEN
450  -          STA (MVDEST),Y
460  -          INY            ;NAECHSTES BYTE
470  -          BNE MLPAGE     ;BIS 256 BYTES VERSCHOBEN SIND
480  -          INC MVSRCE+1   ;NAECHSTE PAGE DER QUELLE
490  -          INC MVDEST+1   ; UND DES ZIELBEREICHES
500  -          DEX            ;PAGEZAEHLER HERUNTERZAEHLEN
510  -          BNE MLPAGE     ;WEITERMACHEN BIS ALLE VOLLEN PAGES FERTIG
520  -MLPART     LDX MVELEN     ;LAENGE DES RESTBEREICHES IN X
530  -          BEQ MLEXIT     ;ZURUECK, WENN REST GLEICH NULL
540  -MLLAST     LDA (MVSRCE),Y ;EIN BYTE VERSCHIEBEN
550  -          STA (MVDEST),Y
560  -          INY            ;NAECHSTES BYTE
570  -          DEX            ;ZAEHLER HERUNTERZAEHLEN
580  -          BNE MLLAST     ;WEITER BIS REST DURCHGEARBEITET IST
590  -MLEXIT     RTS            ;ZURUECK ZUM HAUPTPROGRAMM
600  -;
610  -;**** UP ZUM VERSCHIEBEN MIT UEBERLAPPUNG : MVERHT ****
620  -;
630  -MVERHT     LDA MVELEN+1   ;ZEIGER AUF LETZTE QUELLPAGE RICHTEN
640  -          CLC
650  -          ADC MVSRCE+1
660  -          STA MVSRCE+1   ;FUER DAS MSB DER MAX. QUELLADRESSE
670  -          LDA MVELEN+1   ;ZEIGER AUF LETZTE ZIELPAGE RICHTEN
680  -          CLC
690  -          ADC MVDEST+1
700  -          STA MVDEST+1   ;FUER DAS MSB DER MAX. ZIELADRESSE
710  -          LDY MVELEN     ;LAENGE DES RESTES
720  -          BEQ MRPAGE     ;WENN NULL, DANN NUR GANZE PAGES
730  -MRO        DEY            ;ZAEHLER MINUS EINS
740  -          LDA (MVSRCE),Y ;EIN BYTE VERSCHIEBEN
750  -          STA (MVDEST),Y
760  -          CPY #0         ;REST SCHON FERTIG ?
770  -          BNE MRO        ;WEITER BIS DER GANZE REST VERSCHOBEN IST
780  -MRPAGE     LDX MVELEN+1   ;ANZAHL PAGES ALS ZAEHLER NACH X
790  -          BEQ MREXIT     ;WENN KEINE GANZEN PAGES DANN ENDE
800  -MR1        DEC MVSRCE+1   ;MSB QUELLADRESSE HERUNTERZAEHLEN
810  -          DEC MVDEST+1   ;EBENSO DAS MSB DER ZIELADRESSE
820  -MR2        DEY            ;INDEX Y HERUNTERZAEHLEN
830  -          LDA (MVSRCE),Y ;EIN BYTE VERSCHIEBEN
840  -          STA (MVDEST),Y
850  -          CPY #0         ;ZAEHLER SCHON NULL ?
860  -          BNE MR2        ;WEITER BIS GANZE PAGE VERSCHOBEN IST
870  -          DEX            ;PAGEZAEHLER HERUNTERZAEHLEN
880  -          BNE MR1        ;WEITER BIS ALLE PAGES VERSCHOBEN SIND
890  -MREXIT     RTS            ;ZURUECK ZUM HAUPTPROGRAMM
900  -;
```

Listing 24. »BLOCK« – Programm zum fehlerfreien Verschieben von Speicherinhalten

chenden Kopierteil verzweigen. Genau das tut das nachfolgend vorgestellte Programm BLOCK (Listing 24), welches sowohl auf dem C 64 als auch auf dem C 128 (dort aber nur innerhalb der gerade eingeschalteten Bank) arbeitet. L.A.Leventhal und W.Saville haben das Prinzip 1982 vorgestellt in »6502 Assembly Language Subroutines«. In Bild 15 finden Sie ein Flußdiagramm des Programmes BLOCK:

Zum Ansteuern des Programmes werden drei Vektoren benötigt:

MVELEN ($FA/FB) enthält die Länge des zu verschiebenden Bereiches;

MVDEST ($FC/FD) enthält die Startadresse des Zielbereiches;

MVSRCE ($FE/FF) enthält die Startadresse des Quellbereiches.

Im Hauptprogramm wird zunächst der Abstand der Startadressen von Quell- und Zielbereich berechnet und dieser dann mit der angegebenen Länge des zu verschiebenden Bereiches verglichen. Ist der Abstand kürzer als diese Länge, dann liegt eine Überlappung vor. Es mag Ihnen vielleicht seltsam anmuten, daß das sowohl dann, wenn die Quelle unterhalb, als auch dann, wenn sie oberhalb des

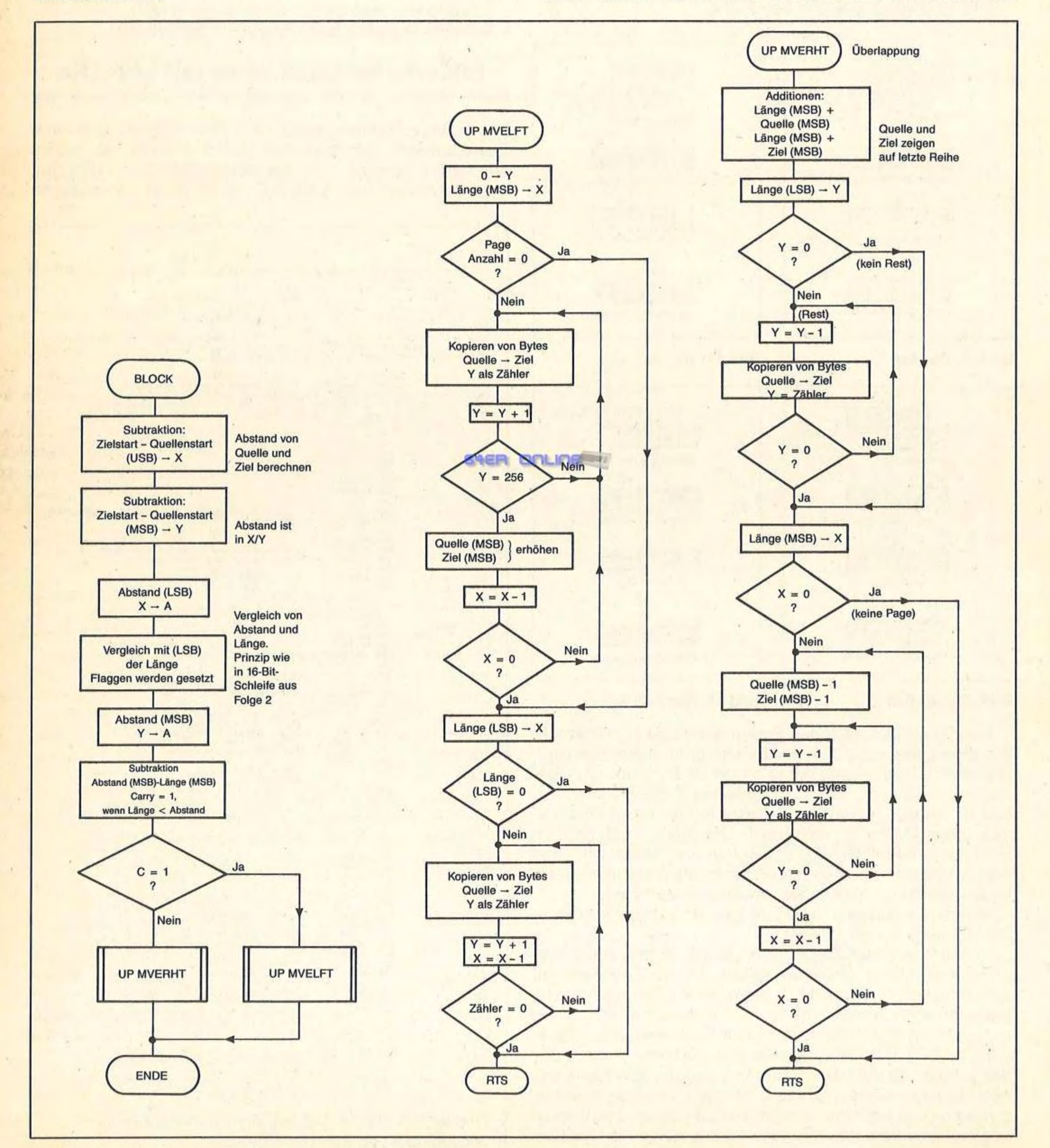


Bild 15. Das Flußdiagramm des Programmes »BLOCK«

Zielbereiches liegt, funktioniert. Das – scheinbare – Geheimnis liegt im Carry-Bit verborgen: Die Routine rechnet automatisch mit Modulo(64K). Ein unter dem Quellbereich liegender Zielbereich erfährt die gleiche Behandlung, als läge er 64K höher. Rechnen Sie diesen Teil mal mit fiktiven Adressen bitweise nach, wenn Sie zu den »Fortgeschrittenen« zu zählen sind.

Der Vergleich des so berechneten Abstandes mit der angegebenen Länge folgt dem gleichen Prinzip, das wir auch schon in der zweiten Folge für Doppelschleifen beliebiger Länge zur Steuerung angewendet haben. Dort haben wir auf diese Weise festgestellt, ob schon die Endadresse erreicht ist. Hier verwenden wir das Verfahren, um herauszubekommen, ob eine Überlappung von Quell- und Zielbereich vorliegt. Ist nämlich die Länge in MVELEN kleiner als der berechnete Abstand, dann finden wir ein gesetztes Carry-Bit vor. Wir haben dann keine Überlappung und verzweigen zur Kopierroutine, die von unten nach oben arbeitet. Durch die Eigenart des vorherigen Umganges mit dem Carry-Bit wird der gleiche Weg auch dann eingeschlagen, wenn eine Überlappung zwar vorliegt, aber der Quell- oberhalb des Zielbereiches liegt.

Den Rest des Programmes bilden die beiden Transportschleifen. MVELFT kopiert aufwärts arbeitend, indem zunächst die ganzen Pages und danach der Rest übertragen wird. MVERHT berechnet zuerst aus der Länge und den beiden MSB der Startadressen (von Quelle und Ziel) die Adresse der letzten Page. Indem ins Y-Register das LSB der Länge gepackt und mittels der indirekt indizierten Adressierung gearbeitet wird, findet hier abwärts zählend als erstes die Übertragung des Restes und danach die der ganzen Pages statt.

Damit Sie BLOCK auf Herz und Nieren prüfen können, finden Sie beiliegend noch ein kleines Basic-Aufrufprogramm namens »BLOCK BAS« (Listing 25).

Ähnlich wie beim Programm BLTUC BAS finden alle Operationen der besseren Verfolgbarkeit wegen im Bildschirmspeicher statt. Auch hier ist wieder – weil ältere C 64-Modelle das benötigen – eine Zeile zum Belegen des Farb-RAM eingefügt worden (Zeile 60), die Sie dann weglassen können, wenn Sie einen neueren C 64 oder einen C 128 verwenden. In der Zeile 90 werden wieder die ersten 40 Zeichen – diesmal in die Bildschirmmitte (ab Speicherstelle 1504) – in den Bildschirmspeicher gePOKEt. Weiterhin haben Sie nun aber die freie Auswahl, wieviele Bytes Sie wohin verschieben möchten. Die Programmzeilen 140 bis 160 übernehmen die Berechnung der MSB und LSB der Adressen und der angegebenen Länge, Zeile 180 schließlich ruft unsere Verschiebe-Routine auf. Viel Spaß beim Ausprobieren!

Speicherinhalte austauschen

Eng verwandt mit den Verschiebe-Routinen und ebenso häufig Thema von Leserfragen ist ein Programminstrument, das es erlaubt, die Inhalte zweier Speicherbereiche auszutauschen. Im Grunde genommen wird ja bei beiden Verschiebeprogrammen (BLTUC und BLOCK) nicht der Inhalt verschoben, sondern nur kopiert. Bei einer Tauschroutine aber verändern sich sowohl der Quell- als auch der Zielbereich.

Überlegen wir uns, wie ein Programm SWAP, das dieses Vertauschen leistet, konstruiert sein muß. Da erhebt sich zunächst wieder die Frage, welche grundsätzlichen Möglichkeiten hier auftreten können. Wieder sind die oben betrachteten Fälle 1 bis 4 ohne Probleme. Die Fälle 5 bis 8 allerdings, die mit Überschneidungen, halte ich hier für sinnlos. Allenfalls dürfte noch ein Fall nützlich sein, in dem

```
10 REM ******* VERSCHIEBEN MIT DEM BLOCK-P
   ROGRAMM *******                          <077>
20 PRINT CHR$(147):REM C 128 = WAIT0,1      <118>
30 POKE 53280,0:POKE 53281,5                <178>
40 POKE 241,1:REM C64 = POKE646,1           <097>
50 REM ----- FARBRAM BELEGEN -----          <122>
60 FOR I=0 TO 1000:POKE 55296+I,1:NEXT I:R
   EM DAUERT EIN WENIG!                     <027>
70 REM ----- BILDSCHIRM BELEGEN -----       <249>
80 S=1504:REM STARTADRESSE QUELLE           <022>
90 FOR I=0 TO 39:POKE S+I,I:NEXT I          <040>
100 REM ----- PARAMETER ABFRAGEN -----      <198>
110 INPUT"WIEVIELE BYTES (SINNVOLL 0 BIS 4
    0)";N                                   <170>
120 INPUT"ZIELORT (SINNVOLL 1024 BIS 1984)
    ";Z                                     <173>
130 REM ----- BERECHNEN UND UEBERGABE ----
    -                                       <012>
140 A=INT(S/256):POKE 255,A:POKE 254,S-256
    *A                                      <038>
150 A=INT(Z/256):POKE 253,A:POKE 252,Z-256
    *A                                      <253>
160 A=INT(N/256):POKE 251,A:POKE 250,N-256
    *A                                      <233>
170 REM ----- VERSCHIEBEN -----             <099>
180 SYS 4864:REM C64 = SYS 49152            <066>
190 END                                     <192>
```

© 64'er

Listing 25. »BLOCK BAS« – Basic-Programm zum Testen von BLOCK

beispielsweise die Endadresse des Bereiches 1 direkt unterhalb der Startadresse des Bereiches 2 liegt, also benachbarte Speicherteile miteinander vertauscht werden. Das kann aber in den Fällen 1 bis 4 erfaßt werden und erfordert daher keine besondere Behandlung.

Somit könnte man das Programm BLOCK als Ausgangsstruktur verwenden. Anstelle des Einsprunges in die Routine für überlappende Bereiche müßten wir eine Routine setzen, die einen darauf aufmerksam macht, daß eine Überschneidung stattfindet. Statt der Sequenzen

```
LDA (V1),Y
STA (V2),Y
```

(V1 und V2 sind die Vektoren, die auf die jeweilige Quell- und Zielbereichsadresse weisen) müßte bei SWAP eine Lösung gefunden werden, die zunächst ein Byte lädt, es dann beiseitelegt, dann aus dem anderen Bereich das entsprechende Byte lädt, dieses dann anstelle des zuerst geladenen speichert, dann das beiseitegelegte wieder hervorholt und an die Stelle des zuletzt geladenen packt.

Zum Beiseitelegen könnte man irgendeine Speicherstelle parat halten. Wir verwenden aber einfach den Stapel:

```
LDA (V1),Y
PHA
LDA (V2),Y
STA (V1),Y
PLA
STA (V2),Y
```

Damit hätten wir es dann. Hier finden Sie nun noch das Programm SWAP (Listing 26) abgedruckt.

Wie Sie sicher erkennen können, haben wir das Programm BLOCK etwas umgeschrieben, nämlich um die eben vorgestellten Teile. Die Meldung, daß eine Überschneidung vorliegt, wird mittels einer kleinen Schleife aus einer Tabelle herausgelesen und durch die Kernel-Routine CHROUT (oder auch BSOUT genannt) auf dem Bildschirm ausgegeben. Diese Routine haben wir schon in der Folge 2 kennengelernt.

Auch diesmal finden Sie anliegend noch ein kleines Basic-Programm (Listing 27), das sich der SWAP-Routine bedient.

Die Belegung des Farb-RAM in Zeile 60 können sich

```
1    -,4,0:1
1    -,4,0:1
10   -;
20   -;
30   -.BASE $1300                ;IN HYPRA-ASS: .BA $C000
40   -;*******************************************
50   -; VERTAUSCHEN ZWEIER SPEICHERBEREICHE (SWAP)
60   -;*******************************************
70   -.DEFINE MVELEN    = $FA    ;IN HYPRA-ASS WIRD STATT DER
80   -.DEFINE MVDEST    = $FC    ; .DEFINE BEFEHLE JEWEILS DER BEFEHL
90   -.DEFINE MVSRCE    = $FE    ; .EQ = $... VERWENDET
100  -.DEFINE PRINT     = $FFD2  ;BILDSCHIRMAUSGABE
110  -;
120  -;IN MVELEN WIRD DIE LAENGE DER ZU VERTAUSCHENDEN BEREICHE ANGEGEBEN
130  -;IN MVDEST DIE STARTADRESSE DES 1. BEREICHES UND IN
140  -;MVSRCE DIE STARTADRESSE DES 2. BEREICHES.
150  -;
160  -;------ PROGRAMM ------------------------------------------
170  -;ALS ERSTES WIRD BESTIMMT, OB DER ZIELBEREICH OBERHALB DES
180  -; QUELLBEREICHES LIEGT UND OB SICH DIE BEIDEN BEREICHE UEBER-
190  -; LAPPEN. EINE UEBERLAPPUNG LIEGT DANN VOR, WENN DIE DIFFERENZ
200  -; VON ZIELADRESSE MINUS QUELLADRESSE KLEINER ALS DIE ANZAHL DER
210  -; ZU VERSCHIEBENDEN BYTES IST.
220  -;
230  -START      LDA MVDEST      ;BERECHNUNG ZIEL MINUS QUELLE
240  -           SEC
250  -           SBC MVSRCE
260  -           TAX
270  -           LDA MVDEST+1
280  -           SBC MVSRCE+1
290  -           TAY
300  -           TXA             ;VERGLEICH MIT LAENGE DES VERSCHIEBEBEREICHES
310  -           CMP MVELEN
320  -           TYA
330  -           SBC MVELEN+1
340  -           BCS DOLEFT      ;VERZWEIGEN, WENN KEINE UEBERLAPPUNG
350  -           JSR MELDEN      ;SONST AUSGABE EINER FEHLERMELDUNG
360  -           JMP EXIT
370  -DOLEFT     JSR MVELFT      ;ZUM UP OHNE UEBERLAPPUNG
380  -EXIT       RTS
390  -;
400  -;**** UP ZUM VERSCHIEBEN OHNE UEBERLAPPUNG: MVELFT ****
410  -;
420  -MVELFT     LDY #0          ;INDEX AUF NULL
430  -           LDX MVELEN+1    ;ANZAHL PAGES IN X
440  -           BEQ MLPART      ;FALLS KEINE GANZEN PAGES DANN REST
450  -MLPAGE     LDA (MVSRCE),Y  ;EIN BYTE LESEN
460  -           PHA             ;SICHERN DES BYTE
470  -           LDA (MVDEST),Y  ;BYTE AUS ANDEREM BEREICH LESEN
480  -           STA (MVSRCE),Y  ;UND UMTRAGEN
490  -           PLA             ;BYTE WIEDER ZURUECKHOLEN
500  -           STA (MVDEST),Y  ;UND UMTRAGEN
510  -           INY             ;NAECHSTES BYTE
520  -           BNE MLPAGE      ;BIS 256 BYTES VERSCHOBEN SIND
530  -           INC MVSRCE+1    ;NAECHSTE PAGE DER QUELLE
540  -           INC MVDEST+1    ; UND DES ZIELBEREICHES
550  -           DEX             ;PAGEZAEHLER HERUNTERZAEHLEN
560  -           BNE MLPAGE      ;WEITERMACHEN BIS ALLE VOLLEN PAGES FERTIG
570  -MLPART     LDX MVELEN      ;LAENGE DES RESTBEREICHES IN X
580  -           BEQ MLEXIT      ;ZURUECK, WENN REST GLEICH NULL
590  -MLLAST     LDA (MVSRCE),Y  ;EIN BYTE LESEN
600  -           PHA             ;SICHERN DES BYTE
610  -           LDA (MVDEST),Y  ;BYTE AUS ANDEREM BEREICH LESEN
620  -           STA (MVSRCE),Y  ;UND UMTRAGEN
630  -           PLA             ;BYTE WIEDER ZURUECKHOLEN
640  -           STA (MVDEST),Y  ;UND UMTRAGEN
650  -           INY             ;NAECHSTES BYTE
660  -           DEX             ;ZAEHLER HERUNTERZAEHLEN
670  -           BNE MLLAST      ;WEITER BIS REST DURCHGEARBEITET IST
680  -MLEXIT     RTS             ;ZURUECK ZUM HAUPTPROGRAMM
690  -;
700  -;**** UP ZUR AUSGABE EINER FEHLERMELDUNG: MELDEN ****
710  -;
720  -MELDEN     LDY #0          ;INDEX AUF NULL
730  -WEITER     LDA TEXT,Y      ;TEXTZEICHEN LADEN
740  -           BEQ ENDE        ;WENN NULLBYTE, DANN ZURUECK ZUM HAUPTPROGRAMM
750  -           JSR PRINT       ;SONST AUF BILDSCHIRM AUSGEBEN
760  -           INY             ;INDEX ERHOEHEN
770  -           JMP WEITER      ;NAECHSTES ZEICHEN AUSGEBEN
780  -ENDE       RTS             ;ZURUECK ZUM HAUPTPROGRAMM
790  -;
800  -TEXT       .BYTE 13        ;HYPRA-ASS:  .BY 13
810  -           .BYTE "UEBERSCHNEIDUNG !";HYPRA-ASS: .TX "UEBERSCHNEIDUNG !"
820  -           .BYTE 13,0      ;HYPRA-ASS: .BY 13,0
830  -;
850  -;
```

Listing 26. »SWAP« – SWAP tauscht Speicherinhalte gegeneinander aus

```
10 REM ****** SWAP TESTPROGRAMM ******         <085>
20 PRINT CHR$(147):REM C 128 = WAITØ,1         <118>
30 POKE 5328Ø,Ø:POKE 53281,5                   <178>
40 POKE 241,1:REM C64 = POKE 646,1             <097>
50 REM ----- FARBRAM BELEGEN (AELTERE C64)
   -----                                       <125>
60 FOR I=Ø TO 1ØØØ:POKE 55296+I,1:NEXT I:R
   EM DAS DAUERT ETWAS                         <048>
70 REM ----- BILDSCHIRM BELEGEN -----          <249>
80 PRINT"WAS DIE ALTE DAME EMPFINDET, WENN
   SIE,{2SPACE}NACHDEM SIE IHREN KANARIEN
   VOGEL "                                     <199>
90 PRINT"GEFUETTERT HAT UND SPAZIEREN GEGA
   NGEN{3SPACE}IST, BEI DER RUECKKEHR DEN
   KAEFIG "                                    <173>
100 PRINT"MIT EINEM LEBENDIGEN TRUTHAHN ZU
   M{7SPACE}PLATZEN VOLL FINDET,"              <125>
110 PRINT CHR$(17)CHR$(17)CHR$(17)CHR$(17)
   CHR$(17)                                    <153>
120 PRINT"ODER DER ALTE HERR,DER,NACHDEM E
   R UEBER NACHT SEINEN KLEINEN TERRIER "      <004>
130 PRINT"AN DIE KETTE GELEGT HAT, EIN NIL
   PFERD{3SPACE}FINDET, DAS UM DIE HUNDHU
   ETTE "                                      <102>
140 PRINT"HERUM SCHNAUBT...":PRINT TAB(2Ø)
   "(LEWIS CARROLL 1882)"                      <133>
150 PRINT CHR$(17)CHR$(17)"JEDER TASTENDRU
   CK FUEHRT ZUM TAUSCH"                       <022>
160 GET A$:IF A$=""THEN 16Ø                    <162>
170 REM ----- PARAMETER FESTLEGEN -----        <016>
180 B1=1Ø24:REM STARTADRESSE BEREICH 1         <133>
190 B2=15Ø4:REM STARTADRESSE BEREICH 2         <023>
200 L =24Ø:REM LAENGE = 6 ZEILEN ZU JE 4Ø
   ZEICHEN                                     <034>
210 REM ----- PARAMETER UEBERGEBEN -----       <035>
220 A=INT(B1/256):POKE 255,A:POKE 254,B1-2
   56*A                                        <051>
230 A=INT(B2/256):POKE 253,A:POKE 252,B2-2
   56*A                                        <192>
240 A=INT(L/256):POKE 251,A:POKE 25Ø,L-256
   *A                                          <181>
250 REM ----- SWAP AUSFUEHREN -----            <001>
260 SYS 4864:REM C64 = SYS 49152               <148>
270 GOTO 16Ø                                   <040>
```

© 64'er

Listing 27. »SWAP BAS« – Basic-Programm, das die Funktion von SWAP überprüft

Besitzer neuerer C 64 und auch des C 128 ersparen, für den alten C 64 ist diese Zeile wichtig. Ab Zeile 80 schreibt das Programm jeweils in die obere und die untere Bildschirmhälfte einen Text. Auf einen Tastendruck werden in den Zeilen 180 bis 240 die Adressen der beiden zu vertauschenden Bereiche und ihre Länge umgerechnet in MSB und LSB und danach in die Abrufspeicherstellen $FA bis $FF gePOKEt. Zeile 260 ruft SWAP auf, Zeile 270 führt den Programmlauf wieder zurück zur Tastaturabfrage in Zeile 160, von wo aus ein erneuter SWAP-Aufruf gestartet wird. Blitzartig wird bei jedem Tastendruck der untere gegen den oberen Text ausgetauscht.

Kombination von BLOCK und SWAP

BLOCK und SWAP sind kurze Routinen, die beide zusammen nur 219 Byte an Platz erfordern. Mit einigem Geschick lassen sich beide auch noch kombinieren. Wenn Sie sicher sind, daß Ihnen nie der Fall unterkommt, der eine Fehlfunktion der BLTUC-Interpreter-Routine verursacht, können Sie sich natürlich auch einer Kombination von BLTUC und SWAP bedienen.

Vielfältige Einsatzmöglichkeiten sind denkbar:

- Bauen Sie doch mal ein Basic-Programm, mit dem Sie einige Hilfsbildschirme (beispielsweise mit Erklärungen zu einem bestehenden Programm) erstellen und mittels BLOCK (oder BLTUC) zum Beispiel ab $C100 abspeichern. Durch Anwendung von SWAP könnten Sie diese Hilfsbildschirme dann gegen den jeweils aktiven Bildschirm austauschen und mit einem zweiten SWAP den normalen Bildschirm wiederherstellen.
- Maschinenprogramme, Hilfsbildschirme und beliebige Speicherinhalte könnten Sie mit BLOCK an Basic-Programme anhängen und mit diesen abspeichern. Beim Laden solcher Kombinationen würde dann durch RUN zunächst BLTUC oder BLOCK aktiviert, das dann diese Anhängsel in die richtigen Speicherteile umlädt.
- Bis zu fünf Bitmaps könnten Sie im Speicher an beliebiger Stelle parat haben und mittels SWAP ohne Verlust von deren Bitmuster in die normale Bitmap blenden.
- Denkbar wäre die Entwicklung einer RAM-Disk, deren Inhalte durch BLOCK und SWAP verwaltet würden.

Sie sehen, daß Schleifen in Assembler auf vielfältige Weise verwendet werden. Wir werden ihnen weiterhin auf Schritt und Tritt begegnen.

Im folgenden Abschnitt erklären wir die beiden wichtigen Zahlensystéme näher.

Was sind Zahlensysteme und wie kommt man mit ihnen zurecht? Bevor wir uns an neue Zahlensysteme wagen, ist

BIT:	7	6	5	4	3	2	1	0	
binär	2^7 = 128	2^6 = 64	2^5 = 32	2^4 = 16	2^3 = 8	2^2 = 4	2^1 = 2	2^0 = 1	dezimal
11111111	1	1	1	1	1	1	1	1	255
10000111	1	0	0	0	0	1	1	1	135

Bild 16. Zur Umwandlung von Binärzahlen in Dezimalzahlen

175:	2	=	87,	Rest	1	→	1	← lsb
87:	2	=	43,	Rest	1	→	1	
43:	2	=	21,	Rest	1	→	1	
21:	2	=	10,	Rest	1	→	1	
10:	2	=	5,	Rest	0	→	0	
5:	2	=	2,	Rest	1	→	1	
2:	2	=	1,	Rest	0	→	0	
1:	2	=	0,	Rest	1	→	1	← msb

ERGEBNIS: 1010 1111 (msb ... lsb)

Bild 17. Die fortlaufende Division durch 2 zur Berechnung einer Binärzahl aus einer Dezimalzahl

es sinnvoll, erst einmal den Aufbau unseres täglich verwendeten zu verstehen. Sie werden sehen, daß man von dieser Basis her alle anderen Systeme begreifen kann.

Dezimalsystem

Haben Sie schon mal kleinen Kindern beim Zählen oder Rechnen zugesehen? Das geht Finger für Finger. Wir besitzen im allgemeinen 10 davon und verwenden daher auch 10 verschiedene Ziffern: 1, 2, 3, 4, 5, 6, 7, 8, 9 und 0. Im Lateinischen heißt zehn decem, weshalb dies die Basis des Dezimalsystems bildet. Solange die Menge dessen, was wir zählen, unter der Basiszahl (also 10) bleibt, haben wir keine Probleme. Was aber kommt nach der 9? Hier hilft nun der geniale Trick weiter, eine Zahl in mehreren Stellen zu schreiben. Ganz rechts außen steht dann die Einerstelle, links daneben eine Zahl, die angibt, wie oft man zu dem Wert in dieser Einerstelle die Basis unseres Zahlensystems (also 10) addieren muß. So bedeutet 49, daß zur Zahl 9 (in der Einerstelle) viermal die Basis 10 zu addieren ist:

$49 = 4 \times 10 + 9$

Irgendwann kommt aber der Moment, wo auch das nicht mehr ausreicht. Was kommt nach 99? Das Konzept mit den unterschiedlichen Stellenwerten läßt sich fortführen: Vor der eben behandelten Zehnerstelle taucht dann die Hunderterstelle auf, die angibt, wie oft zu dem Wert, der sich aus der Einer- und der Zehnerstelle ergibt, das Zehnfache unserer Basis (also 10 x 10 oder 100) zu addieren ist. Als Beispiel sehen wir uns die Zahl 493 an:

$493 = 4 \times 10 \times 10 + 9 \times 10 + 3$

Die nächste vorgelagerte Stelle wäre die Tausenderstelle, die dann angäbe, wie oft zum schon berechneten Rest das Zehnfache des Zehnfachen unserer Basis (10 x 10 x 10 = 1000) zu addieren ist und so weiter. Die Schreibweise ist platzfressend, weshalb man sich der Potenzen bedient. Falls Ihnen dieses Wort nicht geläufig ist: Potenzen sind die Hochzahlen, die angeben, wie oft die Basis mit sich selbst malgenommen wird. So ist:

$100 = 10 \times 10 = 10^2$
$1000 = 10 \times 10 \times 10 = 10^3$

und so fort. Außerdem haben es die Mathematiker als sinnvoll angesehen, festzulegen:

$10 = 10^1$ und $1 = 10^0$

Eine Zahl 24237 kann daher geschrieben werden als:

$24237 = 2 \times 10^4 + 4 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 7 \times 10^0$
$= 2 \times 10000 + 4 \times 1000 + 2 \times 100 + 3 \times 10 + 7$

Die beiden Festlegungen für die Hochzahl 1 und 0 sind übrigens ganz allgemein festgelegt: Eine Zahl hoch 1 ergibt immer die Zahl selbst, eine Zahl hoch 0 ergibt immer 1. Es gilt also:

$2^1 = 2$, $5^1 = 5$ etc.
$2^0 = 1$, $5^0 = 1$ etc.

Das wird uns gleich von Nutzen sein, wenn wir auf andere Zahlensysteme umsteigen. Dieser Trick mit den unterschiedlichen Wertigkeiten der Stellen einer Zahl ist nämlich keinesfalls nur auf das Dezimalsystem beschränkt. Auch bei allen anderen denkbaren Systemen gilt, daß man immer dann, wenn man beim Zählen an die Basis minus 1 herankommt (also 9 im Dezimalsystem), eine nächsthöhere Stelle schafft.

Das Binärsystem

Ein Computer ist - vereinfacht gesehen (für manche mag es wie ein Sakrileg klingen) - im Grunde nur ein Haufen von Schaltern. Von reichlich vielen allerdings und auch sehr kleinen. Jeder Schalter kennt dabei nur zwei Zustände: Ein und Aus. Setzen wir anstelle dieser Worte nun Ziffern ein, dann entspricht dem »Ein« die Ziffer 1, dem »Aus« die Ziffer 0. Zwei Ziffern also: Der Computer befindet sich in der gleichen Lage wie das - bislang noch unentdeckte - Volk der Zweifingerlinge. Weil diese - im allgemeinen - nur zwei Finger besitzen, mit denen sie zählen können, basiert ihr Zahlensystem auf der Zahl zwei. Das lateinische »bini« heißt deutsch »je zwei« und man nennt solch ein System Binärsystem (manchmal auch Dualsystem vom lateinischen »duo«, was »zwei« heißt).

Wie zählen die Zweifingerlinge?

Wie bei uns fangen sie mit der 1 an. Aber das Problem, das uns die auf 9 folgende Zahl bereitet, stellt sich hier schon bei der auf 1 folgenden Zahl. Es gibt ja keine Ziffer 2 in diesem System. Auch die Zweifingerlinge haben vor undenkbaren Zeiten den Trick mit den verschiedenen Stellen herausgefunden. Wenn sie also die auf 1 folgende Zahl schreiben möchten, dann schaffen sie eine neue Stelle, die dann unserer Zehnerstelle entspricht und so fort. Die Zahlen von 1 bis 10 sehen bei den Zweifingerlingen (auch unser Computer ist einer) dann so aus:

Binär	Dezimal	Binär	Dezimal
1	1	110	6
10	2	111	7
11	3	1000	8
100	4	1001	9
101	5	1010	10

Zur Übung können Sie ja mal die Binärzahlen bis 255 aufschreiben. Wenn bei Ihnen dann 255 die Binärzahl 11111111 ergibt, dann haben Sie richtig gezählt.

Unsere Überlegungen von vorhin beim Aufbau des Dezimalsystems helfen uns nun bei der Umrechnung der Binärzahlen in Dezimalzahlen. Die Basis ist hier 2 und eine Binärzahl 1001 kann daher zerlegt werden in:

$1001 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$= 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1$
$= 9$

Daß das stimmt, können Sie in der Tabelle der Zahlen von 1 bis 10 nachprüfen. Auf diese Weise wird die Umrechnung

von Binärzahlen in Dezimalzahlen recht einfach. Als Gedächtnisstütze bedient man sich eines Schemas wie in Bild 16:

In der oberen Reihe finden Sie darin die Bitnummer (das Ganze habe ich auf ein Byte bezogen), darunter die Zweierpotenzen. In den beiden Reihen darunter sind noch zwei Berechnungsbeispiele gezeigt. Die Zweierpotenzen aller Spalten, in denen eine 1 steht, werden addiert und ergeben so den Dezimalwert.

Ebensohäufig stellt sich das Problem anders herum: Aus einer Dezimalzahl soll die Binärzahl berechnet werden. Eine einfache Methode, dies zu tun, ist die fortlaufende Division der Dezimalzahl durch 2. Für Mathe-Spezialisten: Die mod(2)-Funktion (die leider nicht in unseren Basic-Versionen 2.0, 3.5 und 7.0 enthalten ist) wird mehrmals nacheinander auf die Dezimalzahl und die Divisionsergebnisse angewendet. In Bild 17 erkennen Sie das Verfahren:

Jedesmal wird also das Ergebnis der vorangegangenen Division wieder durch 2 geteilt, bis sich 0 ergibt. Die Reste notiert man sich: Sie ergeben in der Reihenfolge »letzte Stelle. . .erste Stelle« die Binärzahl.

Hat man sich erst einmal an die Zahlen der Zweifingerlinge gewöhnt, dann kann man damit ebensogut rechnen wie mit den Dezimalzahlen. Das soll aber an dieser Stelle nicht unser Thema sein. Auch negative Binärzahlen gibt es und solche, die den Dezimalbrüchen (also Zahlen mit Nachkommastellen) entsprechen. All dies können Sie im Kurs »Assembler ist keine Alchimie« in den Kapiteln 11, 13, 29 und 38 nachlesen (der Kurs erschien im 64'er Sonderheft 8/85 komplett, einige Korrekturen dazu wurden im 64'er-Magazin, Ausgabe 4/86, Seite 73, im Fehlerteufelchen veröffentlicht), wo auch auf die Art eingegangen wird, wie unser Computer solche Zahlen verarbeitet. Wir verlassen jetzt das Volk der Zweifingerlinge und suchen ein noch seltsameres auf.

Hexadezimalsystem

Im Mai 1891 entdeckte White das unterirdische Reich Atvabar. Sie werden sich erinnern (oder nicht? Dann lesen Sie es nach im Buch »The Gooddess of Atvabar, being the History of the Discovery of the Interior World and Conquest of Atvabar«, erschienen in New York 1892), daß William R. Bradshaw 1892 über Land und Leute berichtete. Über eines allerdings hat er nichts verlauten lassen, weil es ihn offenbar zu sehr verwirrte: Die Atvabarer sind Sechzehnfingerlinge! Genauso, wie es den Zweifingerlingen schwerfällt, in unserem Dezimalsystem zu rechnen (es fehlen ja sogar die Worte für alle Zahlen, die größer als die Basis 2 sind) hatte White - ein einfacher Seemann - Probleme, die Zahlen der Atvabarer gedanklich zu erfassen, weshalb er das ganze gegenüber Bradshaw einfach verschwieg.

Wir haben diese Schwierigkeiten nicht (oder?) und verwenden anstelle der uns unbekannten Ziffernsymbole einfach die ersten Buchstaben des Alphabets. Wenn solch ein Sechzehnfingerling die Finger an seinen beiden Händen zählt, dann sieht das so aus:

Zählweise																
von uns	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
eines »Atvabarers	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

Auch wenn die Atvabarer ansonsten etwas merkwürdig sind (so fahren sie Fahrräder ohne Räder!), so verwenden sie doch den gleichen Trick bei Zahlen, die größer sind als die Basis minus 1: Auch sie schaffen höherwertige Stellen, wie Sie aus der letzten Zahl der obigen Reihe entnehmen können. Versuchen Sie doch einmal, weiter zu zählen bis 255. Wenn Sie dann auf die Zahl FF kommen, war alles richtig.

Irgendwann einmal in der Anfangszeit der Computerei muß einer der Elektronik-Pioniere etwas von dieser Eigentümlichkeit der Atvabarer erfahren haben. Anders ist es kaum zu erklären, daß das Zahlensystem dieses vergessenen Volkes heute bei Assembler-Programmierern so eine gewichtige Rolle spielt. Eine andere Erklärung wäre es, daß

Hexadezimal	STELLE 3	2	1	0
0	0	0	0	0
1	4096	256	16	1
2	8192	512	32	2
3	12288	768	48	3
4	16384	1024	64	4
5	20480	1280	80	5
6	24576	1536	96	6
7	28672	1792	112	7
8	32768	2048	128	8
9	36864	2304	144	9
A	40960	2560	160	10
B	45056	2816	176	11
C	49152	3072	192	12
D	53248	3328	208	13
E	57344	3584	224	14
F	61440	3840	240	15

Tabelle 2. Umrechnungstabelle von Hexadezimalzahlen in Dezimalzahlen

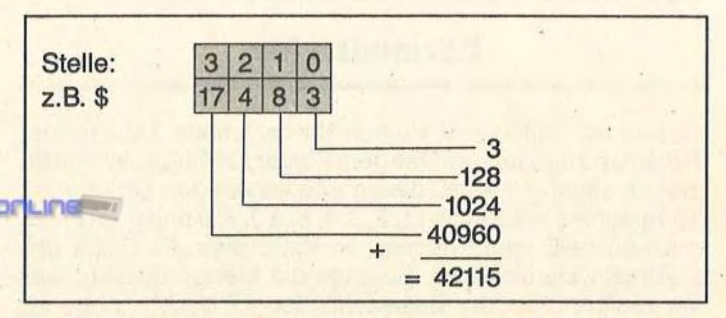


Bild 18. Umrechnung einer Hexadezimalzahl in die Dezimalzahl mittels der Tabelle 2.

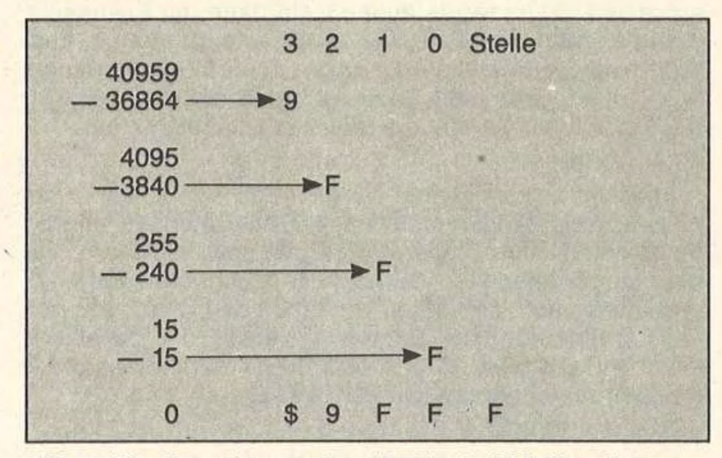


Bild 19. Die Umrechnung einer Dezimalzahl in ihre Entsprechung im Hexadezimalsystem mit Hilfe der Tabelle 1

man im Adreßraum von 8- und 16-Bit-Computern besonders leicht damit rechnen kann. Vielleicht ist das auch nur eine Glaubensfrage.

Wie dem auch sei: Ebenso wie bei anderen Zahlensystemen ist auch dieses hier - auf der Basis 16 (oder F für die Sechzehnfingerlinge) - nach den Regeln aufgebaut, die wir vorhin beim Dezimalsystem erklärt haben. Eine Hexadezimalzahl $831 kann man in die Dezimalzahl umrechnen:

$$\$831 = 8 \times 16^2 + 3 \times 16^1 + 1 \times 16^0$$
$$= 8 \times 256 + 3 \times 16 + 1 \times 1$$
$$= 2097$$

Zwar ist es auf diese Weise möglich, jede Hexzahl umzurechnen; es ist aber auch ziemlich mühsam. Deshalb bedient man sich dazu einer Tabelle, wie sie als Tabelle 2 abgedruckt ist.

Die erste Zeile dieser Tabelle enthält die Stelle der Ziffer, die erste Spalte die Hex-Ziffer. In den Kolonnen sind jeweils die Dezimalzahlen angegeben. Um beispielsweise die Hex-Zahl $A483 in eine Dezimalzahl umzurechnen, geht man vor wie folgt (siehe auch Bild 18):

In der nullten Stelle unserer Zahl steht eine 3. Wir gehen also in die Tabelle und suchen in der Spalte 0/ Zeile 4 (das ist die Zeile, vor der links die 3 steht) den Dezimalwert heraus. Das ist die Zahl 3. Dann gehen wir zur Stelle 1 (das ist die 8 unserer Hex-Zahl). In der Tabelle findet sich in der Spalte 1/ Zeile 9 (die Zeile, vor der 8 steht) der Dezimalwert 128. Den addieren wir zur vorher gefundenen 3 dazu. Für die weiteren Stellen verfahren wir ebenso und erhalten dann - wie im Bild gezeigt - die Dezimalzahl 42115. Probieren Sie dieses Verfahren einmal aus: Nach einiger Zeit wird es Ihnen leichtfallen, auf diese Weise die Zahlenumrechnungen durchzuführen.

Der umgekehrte Weg der Berechnung, nämlich aus einer gegebenen Dezimalzahl die Hex-Zahl zu bestimmen, funktioniert ebenfalls mit der Tabelle ganz gut. (Für die Freaks: Man kann das auch ähnlich wie oben bei den Binärzahlen machen, nämlich mittels einer mod(16)-Funktion). Nehmen wir mal die Dezimalzahl 40959 (siehe Bild 19).

Wir suchen aus den Dezimalwerten der Tabelle den heraus, der gerade noch kleiner ist als unsere Zahl: 36864. Den zu diesem Wert gehörigen Hex-Wert in der linken Randspalte notieren wir uns als die höchste Stelle der Hex-Zahl, ziehen dann den Wert von unserer Dezimalzahl ab und erhalten eine neue dafür: 4095. Wieder suchen wir die nächstkleinere Dezimalzahl aus der Tabelle heraus - das ist nun 3840 - notieren uns den dazugehörigen Hex-Wert, subtrahieren und so fort, wie in Bild 19 gezeigt wurde.

Auch mit den Hexadezimalzahlen kann man natürlich rechnen, nur ähnelt das Rechnen hier einem Alptraum. Es ist auch gar nicht nötig, denn im Gegensatz zu den Binärzahlen kennt unser Computer die Hex-Zahlen gar nicht. Selbst wenn er sie per Monitor oder durch ein geschicktes Programm annimmt, wandelt er sie sogleich wieder in Binärzahlen um. Weshalb dann diese Quälerei mit den Zahlen der Sechzehnfingerlinge? Adressen in 8-Bit-Computern oder Worte in 16-Bit-Computern lassen sich mit genau 16 Bit ausdrücken, was einer Hexzahl von vier Stellen entspricht. Ein Byte ist 8 Bits lang und genau eine zweistellige Hexzahl und ein Halbbyte (Insider nennen sowas ein Nibble) ist 4 Bit lang und läßt sich durch eine einstellige Hexzahl erfassen.

Es kann sogar sein, daß Sie es als Assembler-Programmierer weitgehend schaffen, solchen Zahlensystem-Umrechnungen per Hand fast ganz aus dem Weg zu gehen. Sowohl der Monitor des C 128 als auch beispielsweise der SMON enthalten Funktionen, die diese Umrechnungen für uns erledigen. Die meisten besseren Assembler - so auch der Hypra-Ass - dürfen sowohl mit Hex- als auch mit Dezimalzahlen angesprochen werden, meist wird man hier ohnehin mit symbolischen Adressen oder Werten arbeiten (also solchen, die zu Beginn mittels .EQ einen Namen erhalten haben). Aber wie es der Teufel so will, manchmal vergißt man es, sich die Startadresse eines Programmes rechtzeitig in den Dezimalwert umrechnen zu lassen und möchte nun nicht eigens wieder den Monitor laden oder man findet eine interessante Stelle in einem ROM-Listing, die mal schnell von Basic her ausprobiert werden soll oder...

Genug der Zahlenspiele. Nun beschäftigen wir uns mit Assembler-Programmen, die sich selbst verändern.

Selbstmodifizierende Programme - also Programme, die sich im Verlauf der Abarbeitung selbst verändern - sind dem einen ein Graus, dem andern aber die Essenz der Raffinesse. Welcher Ansicht man auch immer sein mag: Es sind mit dieser Technik recht interessante Dinge möglich, die auf andere Weise nicht oder nur schwer realisierbar wären.

Programme, die sich selbst verändern

Wie unterscheidet unser Computer Programme und Daten? Sehen wir uns zuerst einmal an, wie das in Basic aussieht: Beide (Daten und Programme) werden im RAM streng voneinander getrennt (beim C 128 liegen sie sogar in unterschiedlichen Speicherbänken) und völlig unterschiedlich verwaltet. Deshalb ist die Selbstveränderung von Basic-Programmen auch mit allerlei Tricks verbunden, die entweder über POKEs den Programmspeicher beeinflussen oder im programmierten Direktmodus arbeiten. Ein simples Basic-Beispiel zeigt Listing 28:

```
10 REM *SELBSTMOD. PRG.1*
20 PRINT CHR$(147):I=0
30 PRINT "A";
40 I=I+1:IF I=62 THEN END
50 POKE2112,PEEK(2112)+1
60 GOTO 30
```

Listing 28. Selbsmodifikation mit Hilfe des POKE-Befehls im Basic-Programm

Dieses Programm für den C 64 (bei anderen Computern muß die Adresse in Zeile 50 entsprechend geändert werden) verändert während des Programmablaufes die Speicherstelle 2112. Dort befindet sich der Buchstabe A im PRINT-Befehl der Zeile 30. Durch den POKE-Befehl gelangt nach dem A ein B, dann ein C und so weiter in das PRINT-Argument. Das sehen Sie dann, wenn Sie sich nach dem Ablauf des Programms mit LIST noch einmal die PRINT-Anweisung ansehen: Das A ist verschwunden, statt dessen ist dort ein Grafikzeichen (bei eingeschalteter Groß- und Kleinschreibung) oder der griechische Buchstabe Pi (bei Großschreibung) zu finden. Die andere Technik, also die, die im programmierten Direktmodus arbeitet, bedient sich des Tastaturpuffers. Falls Sie darüber mehr wissen möchten, dann lesen Sie bitte den Artikel »Lernen Sie Ihren Commodore 64 kennen«, Teil 4, in der Ausgabe 8/85 der Zeitschrift Happy-Computer, Seite 45ff. C 128-Benutzer sollten die Ausgabe 7/86 des 64'er-Magazins auf Seite 85 aufschlagen: Dort sind allerlei Verwendungsmöglichkeiten dieser Technik für den großen Bruder des C 64 vorgeführt. Soweit also das Ganze in Basic, wie verhält es sich in Assembler?

Hier existiert für den Computer nur eine lange Straße aufeinanderfolgender Speicherzellen. Der Zentralprozessor orientiert sich am Programmzähler, in dem sich die gerade aktuelle Anschrift befindet. In jeder Hausnummer findet die CPU irgendeinen Code, der sie veranlaßt, darauf zu reagie-

```
10 -.ba $3000
20 .;********************
30 .;* prg.2: selbstmod.*
40 .;********************
50 -;
60 -code     lda #$01       ;buchstabe a
70 -         ldx #401       ;farbe weiss
80 -bild     sta $0400      ;bildschirmspeicher
90 -farb     stx $d800      ;farbram
100 -        inc farb+1
110 -        inc bild+1
120 -        inc code+1
130 -        bne code
140 -        brk
```

Listing 29. Zum Prinzip der Selbstmodifikation in Assemblerprogrammen

ren. Alle derartigen Codes führen zu Veränderungen von Speicherinhalten – und sei es auch nur das Hochzählen des Programmzählers beim NOP-Befehl, das Chaos beim Programmabsturz oder auch das Eintragen von ASCII-Werten in den Bildschirmspeicher. Mal liegen diese Veränderungen weit weg vom Programm-Code, mal näher dran: Nichts hindert uns, auch in dem Speicherteil Änderungen vorzunehmen, in dem das Programm abgelegt ist, was uns mit Assemblern wie dem Hypra-Ass leichtfällt. Listing 29 zeigt, wie man Vergleichbares in Maschinensprache erreichen kann:

Das Programm ist für die älteren Versionen des C 64 geschrieben – daher die Belegung des Bildschirmfarbspeichers –, läuft aber auch auf den anderen Versionen, bei denen man die Zeilen, die sich auf die Farben beziehen, weglassen kann. Erinnern Sie sich bitte an die Art, wie der 6502 und seine kompatiblen Nachkommen Adressen im Speicher ablegen: Wenn wir ein Assemblerprogramm schreiben:

```
STX  $D800
```

dann findet sich im Speicher die Code-Folge:

Speicherstelle	Code	Bedeutung
Farb	8E	Code für absolutes STX
Farb+1	00	LSB der Adresse $D800
Farb+2	D8	MSB der Adresse $D800

Deshalb erhöhen wir Farb + 1 und Bild + 1.

Ebenso wie im Basic-Beispiel zeigt sich auch im Listing 2 ein Nachteil dieser Art der Programmierung: Das Programm kann kein zweites Mal gestartet werden – eben weil wir es verändert haben. Jedenfalls leistet es beim Neustart nicht mehr genau dasselbe. Sehen Sie sich nach dem Programmdurchlauf einmal das Disassemblerlisting an, dann finden Sie in den veränderten Zeilen:

```
CODE      LDA   #$00
BILD      STA   $04FF
FARB      STX   $D8FF
```

Beim Starten dieses veränderten Programms wird zuerst der Klammeraffe (das ist das Zeichen mit dem Code 00) in die Bildschirmspeicherstelle $04FF geschrieben. Erst danach läuft alles seinen gewohnten Gang, weil $FF+1 als $00 verstanden wird. Im Falle dieses Programms hätten wir die Schwierigkeit leicht umgehen können: Wenn wir nämlich anstelle des A mit dem Klammeraffen angefangen hätten, sähe unser Programm nach dem Ablauf genauso aus wie vorher.

Es ist also erforderlich, in solche selbstmodifizierenden Programme einen Reparaturmechanismus einzubauen, der die veränderten Speicherinhalte wieder auf einen definierten Startwert bringt. Das geschieht durch eine Initialisierung vor dem eigentlichen Programm oder durch Rückstellen aller beeinflußten Speicherplätze nach dem Arbeitsteil – was eine weitere Selbstmodifikation wäre. Anstelle des BRK im Listing 29 stünde dann beispielsweise:

```
STX  CODE + 1
DEX
STX  BILD + 1
STX  FARB + 1
BRK
```

Zur Übung können Sie ja mal die andere Möglichkeit – also die Initialisierung vor dem eigentlichen Programm – einbauen.

Anwendung der Selbstmodifikation

Vielleicht haben Sie nun schon eine Vorstellung davon, was für ein mächtiges Programmierinstrument man mit dieser Technik in der Hand hat. Wir haben ja schon im Listing 29 eine Schleife geschrieben und sind dabei ohne die indirekte Adressierung ausgekommen. Der Schritt zur 16-Bit-Schleife ist nun nicht mehr weit: Man veranlaßt einfach, daß nicht nur die LSBs der Adressen (BILD und FARB) anders eingetragen werden, sondern auch die MSBs nach jedem kompletten 8-Bit-Schleifen-Durchlauf. Florian Müller hat sich die Mühe gemacht, in seinem Kurs »Effektives Programmieren in Assembler«, Kapitel 10 (erschienen im Assembler-Sonderheft des 64'er-Magazins, Sonderheft 8/85, Seite 97ff.) allerlei Varianten der Anwendung von Selbstmodifikation in Programmen vorzustellen. Deshalb soll hier nur ein kleiner Überblick gegeben werden.

So ist es beispielsweise möglich, eine ganze Reihe von Befehlen zu simulieren, die es im Sprachschatz des 6502-Assemblers nicht gibt: indirekte JSR-Sprünge (es gibt nur den indirekten JMP-Befehl), indirekte Schiebe-, Dekrementier- und Inkrementierbefehle. Befehle mit unmittelbarer Adressierung (beispielsweise CMP #$20) können veränderliche Argumente erhalten, man kann auf diese Weise beispielsweise den Inhalt des Akku und des X-Registers addieren:

```
    STX  ADD+1  ;X-Register hinter ADC-Befehl
                 ablegen
    ...         ;eventuell weiteres Programm
    CLC         ;Carry-Bit freimachen vor Addition
ADD ADC  #$FF   ;$FF ist nur ein Füllwert (Dummy)
```

Komplette Befehle kann man durch Eintragen des Befehls-Codes umändern, beispielsweise aus einem BCS (Code $B0) ein BCC (Code $90) machen oder Unterprogrammaufrufe verhindern beziehungsweise erlauben (durch Eintragen des Codes für den BIT-Befehl anstelle des JSR-Codes). Ganze Sequenzen lassen sich durch das Programm selbst umschreiben. Sie sehen: Der Möglichkeiten gibt es viele.

Ein kurzer Blick in die CHRGET-Routine

Eine andere Anwendung selbstmodifizierender Programmtechniken befindet sich schon fix und fertig in unserem Computer (hier ist speziell der C 64 gemeint): die sogenannte CHRGET-Routine. Laden Sie doch einmal den SMON und blicken Sie mittels

```
D 0073 008B
```

in den unteren RAM-Bereich hinein. Was Sie dann auf dem Bildschirm sehen, ist dieses kleine Programm, das die Aufgabe hat, den Inhalt des Basic-Speichers Byte für Byte zu lesen und mit bestimmten Markierungen an den Basic-Interpreter zu übergeben. Es handelt sich um eines der wichtigsten Werkzeuge des Interpreters. Wie es genau funktioniert, können Sie nachlesen im Kapitel 25 des

Assembler-Kurses (Sonderheft 8/85, Seite 26), hier würde uns die Besprechung zu weit vom Thema wegführen. Zum Thema aber passen die ersten vier Zeilen:

```
0073     INC $7A
0075     BNE $0079
0077     INC $7B
0079     LDA $0225  ;$0225 steht hier nur als
                     Dummy
007C     ...
```

Wie Sie sicherlich bemerken, steht die Adresse, aus der etwas in den Akku geladen werden soll (Zeile 0079), bei $7A (das LSB) und $7B (das MSB). Was also in der ersten Zeile passiert, ist das Hochzählen der Ladeadresse, die gleich benutzt werden soll. Die nächste Zeile prüft, ob dabei ein Überlauf ($FF+1) stattgefunden hat. In dem Fall ist das Zero-Flag gesetzt, der Sprung nach 0079 findet nicht statt. Zuerst wird noch das MSB der Ladeadresse erhöht. Wie auch immer, die Adresse in $7A/$7B ist nun um 1 größer geworden und der Inhalt der so angezeigten Speicherstelle wird in den Akku geladen.

Bevor wir uns dem zweiten Beispiel zuwenden, noch eine Bemerkung zu einem Nachteil der selbstverändernden Programme: Wie Sie sehen, steht die CHRGET-Routine im RAM – ganz im Gegensatz zur ganzen sonstigen im ROM stehenden Software des C 64. Das hört sich vielleicht trivial an, ist aber schon vorgekommen: Eben weil man aus dem ROM nur lesen, nicht aber hineinschreiben kann, darf auch kein Programm oder auch nur ein Teil davon dort vorhanden sein, das selbstverändernde Techniken benutzt. Wenn Sie EPROMs selbst brennen, sind Sie vielleicht schon einmal über diese Falle gestolpert.

Programmieren einer Befehlserweiterung

Dies ist ein umfangreiches Thema, bei dem wir eine Anwendung der selbstmodifizierenden Programmtechnik kennenlernen, aber auch ein Verfahren, wie man neue Basic-Befehle einbinden kann. Außerdem wird uns eine ganze Palette von Interpreter-Routinen geläufig. Wir werden erstmalig mit Tabellen arbeiten und auch die eben erwähnte CHRGET-Routine bewußt einsetzen. Weil viele Leser wissen wollen, wie man die verschiedenen mathematischen Interpreter-Routinen ansteuert, werden wir dem Basic des C 64 noch einige mathematische Funktionen hinzufügen. Als Listing 30 finden Sie es weiter unten abgedruckt. Listing 31 ist das fertige Programm, das Sie mit dem MSE eingeben müssen.

Vielen Benutzern ist das Basic 2.0 zu dürftig. Auch wenn man nach mathematischen Funktionen sucht, sind es relativ wenige. So stört es beispielsweise, daß man vom gewohnten Gradmaß der Winkel bei Winkelfunktionen wie SIN, COS und TAN abweichen und erst noch auf Bogenmaß umrechnen muß. Außerdem sind es zu wenig Winkelfunktionen und die Umkehrfunktionen (arcus...) sind gar nur in einer einzigen Form vertreten: ATN. Wenn man mit Logarithmen arbeiten möchte, muß man sich immer auf die natürlichen (LOG ist gleich ln) umstellen, statt mit den normalen dekadischen arbeiten zu können. Unser aus zehn Modulen bestehendes Programm erweitert nun das Basic um neun Befehle.

Der erste davon heißt AUS. Damit kann man diese Erweiterung abschalten, falls sie nicht benötigt wird.

Es folgen zwei Funktionen zur Umrechnung von Gradmaß in Bogenmaß und umgekehrt. BOG ermittelt das Bogenmaß eines Winkels:

Aufruf: BOG,Winkel

GRD geht den umgekehrten Weg der Berechnung des Gradmaßes eines im Bogenmaß angegebenen Winkels:

Aufruf: GRD,Winkel

Sind Sie das Rechnen mit dem durch LOG erzeugten natürlichen Logarithmus leid, dann verwenden Sie DLGR für den normalen dekadischen Logarithmus:

Aufruf: DLGR,Argument

Bei den trigonometrischen Funktionen steht Ihnen nun neben SIN, COS und TAN auch der Kotangens COT zur Verfügung:

Aufruf: COT,Winkel im Bogenmaß

Die bislang nur durch recht komplizierte Formeln zu ermittelnden Umkehrfunktionen (die im Handbuch sogar teilweise falsch angegeben sind) des Sinus, Cosinus und Kotangens erreichen Sie durch die nächsten drei Funktionen ARCS, ARCC und ACOT:

Aufruf: ARCS,Argument
ARCC,Argument
ACOT,Argument

Ein kleines Bonbon noch am Schluß: Ein Polynom ist ein Ausdruck der Form:

$y = a_0 + a_1x + a_2x^2 + a_3x^3 + ... a_nx^n$

Dabei ist n der Grad des Polynoms. Die einzelnen »a« nennt man Koeffizienten. Durch den neuen Befehl POLY kann solch ein Polynom schnell berechnet werden, indem man angibt, für welchen Wert »x« man die Berechnung ausführt, welchen Grad das Polynom hat und wie die Koeffizienten heißen:

Aufruf: POLY,x,n,a_n,a_{n-2},...,a_1,a_0

Solche Polynome spielen in vielen Bereichen der Mathematik und der Statistik eine wichtige Rolle.

Noch zu einer Besonderheit all dieser Funktionen, die ihren Aufruf betrifft. Bei der Beschreibung des ersten Moduls werden Sie sehen, daß die hier gewählte Methode der Befehlserweiterung relativ einfach ist. Das bietet zwar den Vorzug (der im Rahmen dieses Kurses erst einmal Vorrang genießt), daß man leicht verstehen kann, wie das Ganze funktioniert, hat aber in der Handhabung der neuen Befehle einige Nachteile. Ein Nachteil betrifft die Ausgabe der durch die neuen Funktionen ermittelten Ergebnisse. Während man beim Sinus beispielsweise gewohnt ist, A=SIN(x) oder B=SQR(SIN(x)) zu schreiben, die Funktion selbst also wie einen Variablenwert verwenden kann, geht das bei unseren Funktionen nicht. Das hätte einen tieferen Eingriff in die Interpreterschleife erfordert. Andererseits war es uns zu primitiv, lediglich das Ergebnis nach der Berechnung auf dem Bildschirm ausgeben zu lassen: Man sollte schon damit weiterrechnen können. Der Kompromiß sieht etwas merkwürdig aus, funktioniert aber (und später, wenn wir weitere Formen der Befehlserweiterung kennengelernt haben, können wir das Programm auch umbauen). Das Ergebnis steht immer in der Variablen, die als letzte vor dem Funktionsaufruf genannt worden ist. Um also in der Variablen A das Bogenmaß eines Winkels zu speichern, ruft man auf:

```
A=A:BOG,Winkel
```

oder um den dekadischen Logarithmus eines Ausdruckes in A abzulegen beispielsweise:

```
A=0:DLGR,SQR(x)
```

Wenn man zwei von unseren neuen Funktionen nacheinander verwendet, beispielsweise hier den Kotangens eines Winkels, der zuvor ins Bogenmaß umgerechnet wurde, dann kann man schreiben:

```
B=0:BOG,Winkel:A=A:COT,B
```

Es ist aber auch ein abgekürzter Weg möglich, denn ein interner Zeiger weist weiterhin auf die bezeichnete Variable:

```
A=A:BOG,Winkel:COT,A
```

In beiden Fällen steht hinterher der Ergebniswert in der

```
10   -;*****************************
20   -;*                           *
30   -;*   PROGRAMM 3 / MODUL 1    *
40   -;*     Erweiterung der       *
50   -;*    Interpreterschleife    *
60   -;*                           *
70   -;*  Heimo Ponnath  HH  1986  *
80   -;*                           *
90   -;*****************************
100  -;
110  -            .ba $5000
120  -;
130  -;------ Labels ---------------
140  -;
150  -            .eq forpnt=$49 ;Variablenzeiger
160  -            .eq chrget=$73 ;chrget-Routine
170  -            .eq chrgot=$79 ;chrgot-Routine
180  -            .eq txtptr=$7a ;chrget-Zeiger
190  -;
200  -            .eq igone=$0308;Vektor zum Routinenaufruf
210  -;
220  -            .eq error=$a437;Fehlermeldung und READY
230  -            .eq newstt=$a7ae;interpreterschleife
240  -            .eq gone1=$a7e4;alter Inhalt von igone
250  -            .eq intend=$a7e7;Ende interpreterschleife
260  -            .eq frmnum=$ad8a;Numerischen Wert einlesen
270  -            .eq chkcom=$aefd;Komma ueberlesen
280  -            .eq facinx=$b1aa;FAC zu Integer in Y/A
290  -            .eq getbytc=$b79b;Byte in X-Register einlesen
300  -            .eq fsub=$b850 ;FAC=Mem-FAC
310  -            .eq eins=$b9bc ;das ist 1
320  -            .eq log=$b9ea  ;FAC=log(FAC)
330  -            .eq fmult=$ba28;FAC=FAC*Mem
340  -            .eq fdiv=$bb0f ;FAC=Mem/FAC
350  -            .eq movfm=$bba2;Mem in FAC
360  -            .eq movmf=$bbd4;FAC in Speicher
370  -            .eq abs=$bc58  ;FAC=abs(FAC)
380  -            .eq fcomp=$bc5b;Vergleich FAC mit Mem
390  -            .eq sqr=$bf71  ;FAC=sqr(FAC)
400  -            .eq polyx=$e059;Polynomauswertung
410  -            .eq cos=$e264  ;FAC=cos(FAC)
420  -            .eq sin=$e26b  ;FAC=sin(FAC)
430  -            .eq pihalb=$e2e0;das ist Pi/2
440  -            .eq atn=$e30e  ;FAC=atn(FAC)
450  -;
460  -            .eq polyvar=polytab-4
470  -;
480  -;------ Initialisierung -------
490  -;
500  -init       lda #<(start)  ;lsb eigene Routine
510  -           sta igone      ;in vektor schreiben
520  -           lda #>(start)  ;msb
530  -           sta igone+1
540  -           rts
550  -;
560  -;------ Abschalten ------------
570  -;
580  -aus        lda #<(gone1)  ;vektor auf
590  -           sta igone      ;Normalwert
600  -           lda #>(gone1)  ;zurueckstellen
610  -           sta igone+1
620  -           rts
630  -;
640  -;-Erweiterte Interpreterschleife-
650  -;
660  -start      jsr chrget     ;Zeichen holen
670  -           cmp #$60       ;Buchstabe?
680  -           bcs ende       ;Basic-Code
690  -           cmp #$41       ;Buchstabe A ?
700  -           bcc ende       ;Sonderzeichen
710  -           sta akku       ;Akku sichern
720  -           ldx #$00
730  -           stx befnr      ;Befehlsnr. auf 0
740  -int1       ldy #$00
750  -           inc befnr      ;Befehlsnr. + 1
760  -           lda beftab,x   ;Zeichen aus Befehlstabelle
770  -           bne int2       ;kein Trennzeichen
780  -;
790  -           lda akku       ;Zurueck ins
800  -ende       jsr chrgot     ;normale Basic
810  -           jmp intend     ;springen
820  -;
830  -;--- Adresse suchen -----------
840  -;
850  -int2       cmp (txtptr),y ;Vergleich mit Basictext
860  -           bne rest       ;ungleich
870  -           iny            ;Basictextindex+1
880  -           inx            ;Befehltab.-Index+1
890  -           lda beftab,x   ;naechstes Zeichen
900  -           bne int2       ;pruefen
910  -           clc
920  -           tya            ;Befehlsindex um
930  -           adc txtptr     ;Befehlslaenge
940  -           sta txtptr     ;erhoehen
950  -           bcc lab1       ;Uebertrag?
960  -           inc txtptr+1   ;msb erhoehen
970  -lab1       lda befnr      ;Befehlsnr.
980  -           asl            ;verdoppeln
990  -           tax            ;und als Index in
1000 -           lda sprtab,x   ;Sprungtabelle
1010 -           sta sprung+1   ;lsb Sprung
1020 -           lda sprtab1,x  ;msb lesen
1030 -           sta sprung+2   ;msb
1040 -;
1050 -;-- Selbstmodifizierender Teil --
1060 -;
1070 -sprung     jsr $ffff      ;Dummy
1080 -;
1090 -;-- Zurueck zum Interpreter -----
1100 -;
1110 -           jmp ende
1120 -;
1130 -;-restl. Befehlstext ueberlesen--
1140 -;
1150 -rest       inx
1160 -           lda beftab,x
1170 -           bne rest       ;bis Trennzeichen
1180 -           inx
1190 -           jmp int1       ;naechster Befehl
1200 -;
1210 -;
1220 -;********************************
1230 -;*                              *
1240 -;*     Programm 3  Modul 2      *
1250 -;* Umrechnung in Bogenmass (BOG) *
1260 -;*                              *
1270 -;********************************
1280 -;
1290 -bog        lda forpnt       ;Variablenzeiger auf Stapel
1300 -           pha
1310 -           lda forpnt+1
1320 -           pha
1330 -           jsr chkcom       ;Komma pruefen
1340 -           jsr frmnum       ;Numerischen Ausdruck holen
1350 -           lda #<(bogfak)   ;Faktor Pi/180
1360 -           ldy #>(bogfak)
1370 -           jsr fmult        ;Multiplikation
1380 -           pla              ;x/y auf Variable
1390 -           tay
1400 -           pla
1410 -           tax
1420 -           jsr movmf        ;FAC in Variable
1430 -           rts
1440 -;
1450 -;
1460 -;********************************
1470 -;*                              *
1480 -;*     Programm 3  Modul 3      *
1490 -;* Umrechnung in Gradmass (GRD) *
1500 -;*                              *
1510 -;********************************
1520 -;
1530 -grd        lda forpnt       ;Variablenzeiger auf Stapel
1540 -           pha
1550 -           lda forpnt+1
1560 -           pha
1570 -           jsr chkcom       ;Komma pruefen
1580 -           jsr frmnum       ;Numerischen Ausdruck holen
1590 -           lda #<(grdfak)   ;Faktor 180/Pi
1600 -           ldy #>(grdfak)
1610 -           jsr fmult        ;Multiplikation
1620 -           pla              ;x/y auf Variable
1630 -           tay
1640 -           pla
1650 -           tax
1660 -           jsr movmf        ;FAC in Variable
1670 -           rts
1680 -;
1690 -;
1700 -;********************************
1710 -;*                              *
1720 -;*     Programm 3  Modul 4      *
1730 -;* Dekadischer Logarithmus(DLGR) *
1740 -;*                              *
1750 -;********************************
1760 -;
1770 -dlog       lda forpnt       ;Variablenzeiger auf Stapel
1780 -           pha
1790 -           lda forpnt+1
1800 -           pha
1810 -           jsr chkcom       ;Komma pruefen
1820 -           jsr frmnum       ;Numerischen Ausdruck holen
1830 -           jsr log          ;Logarithmieren
1840 -           lda #<(logfak)   ;Faktor 1/ln10
1850 -           ldy #>(logfak)
1860 -           jsr fmult        ;Multiplikation
1870 -           pla              ;x/y auf Variable
1880 -           tay
1890 -           pla
1900 -           tax
1910 -           jsr movmf        ;FAC in Variable
1920 -           rts
1930 -;
1940 -;
1950 -;********************************
1960 -;*                              *
1970 -;*     Programm 3  Modul 5      *
1980 -;*   Kotangensfunktion (COT)    *
1990 -;*                              *
2000 -;********************************
2010 -;
2020 -cot        lda forpnt       ;Variablenzeiger auf Stapel
2030 -           pha
2040 -           lda forpnt+1
2050 -           pha
2060 -           jsr chkcom       ;Komma pruefen
2070 -           jsr frmnum       ;Numerischen Ausdruck holen
2080 -           ldx #<(zwsp1)    ;und beiseite legen
2090 -           ldy #>(zwsp1)
2100 -           jsr movmf
2110 -           jsr cos          ;Cosinus bilden
2120 -           ldx #<(zwsp2)    ;und sichern
2130 -           ldy #>(zwsp2)
2140 -           jsr movmf
2150 -           lda #<(zwsp1)    ;Wert zurueckholen
2160 -           ldy #>(zwsp1)
2170 -           jsr movfm
2180 -           jsr sin          ;Sinus bilden
2190 -           lda #<(zwsp2)    ;Division
2200 -           ldy #>(zwsp2)    ;FAC=zwsp2/FAC
2210 -           jsr fdiv
2220 -           pla              ;x/y auf Variable
2230 -           tay
2240 -           pla
2250 -           tax
2260 -           jsr movmf        ;FAC in Variable
2270 -           rts
2280 -;
2290 -;
2300 -;********************************
2310 -;*                              *
2320 -;*     Programm 3  Modul 6      *
2330 -;* Arcuscotangensfunktion (ACOT) *
2340 -;*                              *
2350 -;********************************
2360 -;
```

```
2370 -acot      lda forpnt     ;Variablenzeiger auf Stapel
2380 -          pha
2390 -          lda forpnt+1
2400 -          pha
2410 -          jsr chkcom     ;Komma pruefen
2420 -          jsr frmnum     ;Numerischen Ausdruck holen
2430 -          jsr atn        ;Arcustangens bilden
2440 -          lda #<(pihalb) ;Zeiger auf Pi/2
2450 -          ldy #>(pihalb)
2460 -          jsr fsub       ;FAC=pihalb-FAC
2470 -          pla            ;x/y auf Variable
2480 -          tay
2490 -          pla
2500 -          tax
2510 -          jsr movmf      ;FAC in Variable
2520 -          rts
2530 -;
2540 -;
2550 -;*********************************
2560 -;*                               *
2570 -;*      Programm 3  Modul 7      *
2580 -;*   Arcussinusfunktion (ARCS)   *
2590 -;*                               *
2600 -;*********************************
2610 -;
2620 -asin      lda forpnt     ;Variablenzeiger auf Stapel
2630 -          pha
2640 -          lda forpnt+1
2650 -          pha
2660 -          lda #$00       ;Flagge auf Null
2670 -          sta flag       ;setzen
2680 -          jsr chkcom     ;Komma pruefen
2690 -          jsr frmnum     ;Numerischen Ausdruck holen
2700 -easin     ldx #<(zwsp1)  ;und sichern
2710 -          ldy #>(zwsp1)
2720 -          jsr movmf
2730 -          jsr abs        ;Absolutwert berechnen
2740 -          lda #<(eins)   ;Vergleich mit
2750 -          ldy #>(eins)   ;Fliesskommawert
2760 -          jsr fcomp      ;von 1
2770 -          beq argok      ;gleich 1
2780 -          rol            ;Bit 7 in Carry
2790 -          bcs argok      ;kleiner 1
2800 -          pla            ;Stapel leeren
2810 -          pla
2820 -          ldx #$0e       ;Fehlernummer
2830 -          jmp error      ;Fehler und Ready
2840 -argok     lda #<(zwsp1)  ;Wert zurueck
2850 -          ldy #>(zwsp1)  ;in FAC
2860 -          jsr movfm
2870 -          lda #<(zwsp1)  ;Bilden von
2880 -          ldy #>(zwsp1)  ;x*x
2890 -          jsr fmult      ;FAC=x↑2
2900 -          lda #<(eins)   ;Bilden von
2910 -          ldy #>(eins)   ;1-FAC
2920 -          jsr fsub       ;FAC=1-x↑2
2930 -          jsr sqr        ;FAC=SQR(1-x↑2)
2940 -          lda #<(zwsp1)  ;Bilden von
2950 -          ldy #>(zwsp1)  ;x/FAC
2960 -          jsr fdiv       ;FAC=x/SQR(1-x↑2)
2970 -          jsr atn        ;FAC=ASIN!
2980 -          lda flag       ;Flagge pruefen
2990 -          bne retour     ;zurueck zu ACOS
3000 -          pla            ;in Variable
3010 -          tay            ;schreiben
3020 -          pla
3030 -          tax
3040 -          jsr movmf      ;FAC in Variable
3050 -retour    rts
3060 -;
3070 -;
3080 -;*********************************
3090 -;*                               *
3100 -;*      Programm 3  Modul 8      *
3110 -;*  Arcuscosinusfunktion (ARCC)  *
3120 -;*                               *
3130 -;*********************************
3140 -;
3150 -acos      lda forpnt     ;Variablenzeiger auf Stapel
3160 -          pha
3170 -          lda forpnt+1
3180 -          pha
3190 -          lda #$ff       ;Flagge auf 255
3200 -          sta flag       ;setzen
3210 -          jsr chkcom     ;Komma pruefen
3220 -          jsr frmnum     ;Numerischen Ausdruck holen
3230 -          jsr easin      ;Berechnen des asin
3240 -          lda #<(pihalb) ;Bilden der
3250 -          ldy #>(pihalb) ;Differenz
3260 -          jsr fsub       ;FAC=pihalb-asin=acos
3270 -          lda #$00       ;Flagge zurueckstellen
3280 -          sta flag
3290 -          pla            ;in Variable
3300 -          tay            ;schreiben
3310 -          pla
3320 -          tax
3330 -          jsr movmf      ;FAC in Variable
3340 -          rts
3350 -;
3360 -;
3370 -;*********************************
3380 -;*                               *
3390 -;*      Programm 3  Modul 9      *
3400 -;*  Polynomberechnung    (POLY)  *
3410 -;*                               *
3420 -;*********************************
3430 -;
3440 -poly      lda forpnt     ;Variablenzeiger auf Stapel
3450 -          pha
3460 -          lda forpnt+1
3470 -          pha
3480 -          jsr chkcom     ;Komma pruefen
3490 -          jsr frmnum     ;Numerischen Ausdruck holen
3500 -          ldx #<(zwsp1)  ;und sichern
3510 -          ldy #>(zwsp1)
3520 -          jsr movmf
3530 -          jsr chkcom     ;Polynomgrad
3540 -          jsr frmnum     ;naechsteZahlholen
3550 -          jsr facinx     ;in Integer wandeln in Y/A
3560 -          sty polytab    ;und ablegen
3570 -          iny            ;Koeffizientenzahl
3580 -m0        sty flag       ;sichern
3590 -          clc            ;Addieren
3600 -          lda m1+1       ;von 5 zur
3610 -          adc #$05       ;Ablegeadresse
3620 -          sta m1+1
3630 -          lda m2+1
3640 -          adc #$00
3650 -          sta m2+1
3660 -          jsr chkcom     ;naechster
3670 -          jsr frmnum     ;Koeffizient
3680 -m1        ldx #<(polyvar);lsb Zieladresse
3690 -m2        ldy #>(polyvar);msb
3700 -          jsr movmf      ;ablegen
3710 -          ldy flag       ;Zaehler laden
3720 -          dey
3730 -          bne m0         ;noch Koeffizienten?
3740 -          lda #<(polyvar);restaurieren
3750 -          sta m1+1       ;der Zieladresse
3760 -          lda #>(polyvar)
3770 -          sta m2+1
3780 -          lda #<(zwsp1)  ;Argument
3790 -          ldy #>(zwsp1)  ;zurueck
3800 -          jsr movfm      ;in FAC
3810 -          lda #<(polytab);Aufruf
3820 -          ldy #>(polytab);der Routine
3830 -          jsr polyx      ;FAC=POLY(x)
3840 -          pla            ;in Variable
3850 -          tay            ;schreiben
3860 -          pla
3870 -          tax
3880 -          jsr movmf      ;FAC in Variable
3890 -          rts
3900 -;
3910 -;
3920 -;*********************************
3930 -;*                               *
3940 -;*   Programm 3 Tabellenmodul    *
3950 -;*   Tabellen und Hilfszellen    *
3960 -;*                               *
3970 -;*********************************
3980 -;
3990 -;------ Konstanten ------------
4000 -;
4010 -bogfak    .by $7b,$0e,$fa,$35,$0f;Pi/180
4020 -grdfak    .by $86,$65,$2e,$e0,$d2;180/Pi
4030 -logfak    .by $7f,$5e,$5b,$d8,$aa;1/ln10
4040 -zwsp1     .by $00,$00,$00,$00,$00;Zwischenspeicher 1
4050 -zwsp2     .by $00,$00,$00,$00,$00;und 2
4060 -;------ Hilfszellen -----------
4070 -;
4080 -befnr     .by $00             ;Befehlsnummer
4090 -akku      .by $00             ;Zw'Speicher f. Akku
4100 -flag      .by $00             ;Marke fuer ACOS, Zaehler fuer POLY
4110 -;
4120 -;---- Sprungtabelle -------------
4130 -;
4140 -sprtab    .by $e7
4150 -sprtab1   .by $a7
4160 -          .wo aus
4170 -          .wo bog
4180 -          .wo grd
4190 -          .wo dlog
4200 -          .wo cot
4210 -          .wo acot
4220 -          .wo asin
4230 -          .wo acos
4240 -          .wo poly
4250 -          .by 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
4260 -;
4270 -;---- Befehlstabelle ------------
4280 -;
4290 -beftab    .tx "aus"
4300 -          .by 0
4310 -          .tx "bog"
4320 -          .by 0
4330 -          .tx "grd"
4340 -          .by 0
4350 -          .tx "dlgr"
4360 -          .by 0
4370 -          .tx "cot"
4380 -          .by 0
4390 -          .tx "acot"
4400 -          .by 0
4410 -          .tx "arcs"
4420 -          .by 0
4430 -          .tx "arcc"
4440 -          .by 0
4450 -          .tx "poly"
4460 -          .by 0,0
4470 -;
4480 -;Hiernach noch Platz fuer weitere
4490 -;8 Befehlstexte lassen.
4500 -;
4510 -          .by 0,0,0,0,0,0,0,0,0,0
4520 -          .by 0,0,0,0,0,0,0,0,0,0
4530 -          .by 0,0,0,0,0,0,0,0,0,0
4540 -          .by 0,0,0,0,0,0,0,0,0,0
4550 -          .by 0,0,0,0,0,0,0,0,0,0
4560 -          .by 0,0,0,0,0,0,0,0,0,0
4570 -;
4580 -;--- Tabelle fuer Polynome ---
4590 -;
4600 -polytab   .by 0,0,0,0,0,0,0,0,0,0
4610 -          .by 0,0,0,0,0,0,0,0,0,0
4620 -          .by 0,0,0,0,0,0,0,0,0,0
4630 -          .by 0,0,0,0,0,0,0,0,0,0
4640 -          .by 0,0,0,0,0,0,0,0,0,0
4650 -          .by 0,0,0,0,0,0,0,0,0,0
4660 -          .by 0,0,0,0,0,0,0,0,0,0
4670 -          .by 0,0,0,0,0,0,0,0,0,0
4680 -;
```

Listing 30. Eine Basic-Befehlserweiterung in 10 Modulen, die einige mathematische Befehle ergänzt

Variablen A, die nun ganz normal weiterverwendet werden kann.

Wie startet man diese Erweiterung? Das kommt ganz darauf an, wohin Sie sie im Speicher legen. Im Modul 1 wurde in Zeile 110 willkürlich der Start nach $5000 gelegt, was den Start durch SYS 20480 ermöglicht. Falls Sie diesem Vorschlag folgen, oder die Erweiterung statt nach $C000 (dann erfolgt der Start durch SYS 49152) in den Basic-Speicherraum legen, dann achten Sie bitte darauf, den betreffenden Speicherbereich vor dem Überschreiben durch Basic-Text, Variable oder Strings zu schützen.

Modul 1 unseres Programmes

Sehen wir uns zunächst die Label an, die im gesamten Programm benutzt werden. Die ersten fünf Adressen sind Zeiger in der Form LSB/MSB, von denen hier immer nur die niedrigere Adresse genannt wird, weil man im Programm mit LABEL und LABEL+1 arbeiten kann. Genaue Beschreibungen dieser Vektoren finden Sie im Kurs »Memory Map« von Dr. Hauck (komplett veröffentlicht im Sonderheft 7/86 des 64'er-Magazins). Deshalb soll hier nur eine kurze Erläuterung dieser Vektoren folgen.

Die Betriebssystemroutine ORPNT ist für uns nicht interessant, weil auch immer mit einem Zeiger auf die zuletzt angesprochene Variable gearbeitet wird. Erinnern Sie sich an unsere etwas ungewöhnliche Ausgabeform? Dazu brauchen wir diesen Zeiger.

CHRGET, CHRGOT und TXTPTR gehören alle zur CHRGET-Routine und dienen dazu, das jeweils nächste Byte aus dem Basic-Programmtext zu holen und zu identifizieren. Hauck beschreibt diese Funktion recht gut.

Auch IGONE ist von Hauck erklärt worden. Das ist ein Zeiger, der normalerweise nach $A7E4 zeigt (von uns in Zeile 240 als GONE1 bezeichnet) und für die Auswertung des Basic-Textes bedeutsam ist. Wir verbiegen diesen Vektor auf unser eigenes Programm, zu dem wir noch kommen werden.

Die folgenden Adressen sind Interpreter-Routinen, die wir uns nutzbar machen, meist solche mathematischer Art. Dazu noch einige Anmerkungen: Wir werden in einer späteren Folge noch genau auf die sogenannten Fließkommazahlen, ihre verschiedenen Speicherformate und die beiden Fließkomma-Akkumulatoren FAC und ARG eingehen. Die übliche Art der Zahlenverarbeitung im C 64 (und auch im C 128) ist die Verarbeitung im Fließkommaformat. Dabei spielt der sogenannte Fließkomma-Akkumulator 1, der allgemein FAC genannt wird und der in den Speicherstellen $61 bis $66 steht, eine ähnlich zentrale Rolle wie der Akkumulator bei den einfachen Assembler-Programmen. Die meisten mathematischen Routinen erwarten das Argument im FAC und geben das Ergebnis im FAC aus. Manchmal ist die Verwendung eines Hilfsakkumulators sinnvoll, der sogenannte ARG ($69 und $6E). Es gibt im Prinzip zwei Formate für Fließkommazahlen in unserem Computer: Als FLPT-Format bezeichne ich die Speicherung der Daten im FAC und ARG in 6 Byte, als MFLPT-Format die im normalen Speicherraum, die nur 5 Byte beansprucht.

Damit ergibt sich die Notwendigkeit folgender Routinen:
1) Routinen, die Werte als Zahlen, Variable oder mathematische Ausdrücke aus dem Basic-Text lesen, ins FLPT-Format bringen und im FAC ablegen.
2) Routinen, die Zahlen aus dem FAC in den normalen Speicher transportieren und dabei die Übersetzung ins MFLPT-Format leisten und Routinen, die den umgekehrten Weg gehen.
3) Routinen, die die nötigen mathematischen Operationen an der Zahl ausführen, die im FAC steht und das Ergebnis im FAC ablegen.
4) Routinen, die dasselbe wie in 3) ausgedrückt leisten, dazu aber noch weitere Zahlen verwenden, die im normalen Speicherraum im MFLPT-Format vorhanden sind.

Wenden wir uns nun den in Listing 30 definierten Labels zu, die Interpreter-Einsprungadressen definieren. (All diese Interpreter-Routinen existieren einsatzbereit in unserem Computer. Eine Übersicht finden Sie beispielsweise im Assembler-Sonderheft 8/85 ab Seite 178). Sehen wir sie uns der Reihe nach an:

ERROR
Ausgabe von Fehlermeldungen und READY-Status
- $A437 dezimal 42039
- Fehlernummer in X-Register

Alle weiteren Angaben, die Sie bei den meisten anderen Routinen finden (benutzte Speicherstellen und Register,

```
Name : modul 1-10              5000 52fd
-----------------------------------------
5000 : a9 16 8d 08 03 a9 50 8d   f3
5008 : 09 03 60 a9 e4 8d 08 03   c1
5010 : a9 a7 8d 09 03 60 20 73   ac
5018 : 00 c9 60 b0 19 c9 41 90   31
5020 : 15 8d 20 52 a2 00 8e 1f   f1
5028 : 52 a0 00 ee 1f 52 bd 46   b0
5030 : 52 d0 09 ad 20 52 20 79   ea
5038 : 00 4c e7 a7 d1 7a d0 28   d2
5040 : c8 e8 bd 46 52 d0 f5 18   68
5048 : 98 65 7a 85 7a 90 02 e6   e4
5050 : 7b ad 1f 52 0a aa bd 22   e5
5058 : 52 8d 63 50 bd 23 52 8d   ad
5060 : 64 50 20 ff ff 4c 36 50   d0
5068 : e8 bd 46 52 d0 fa e8 4c   2c
5070 : 29 50 a5 49 48 a5 4a 48   bf
5078 : 20 fd ae 20 8a ad a9 06   0f
5080 : a0 52 20 28 ba 68 a8 68   b9
5088 : aa 20 d4 bb 60 a5 49 48   d8
5090 : a5 4a 48 20 fd ae 20 8a   5b
5098 : ad a9 0b a0 52 20 28 ba   2d
50a0 : 68 a8 68 aa 20 d4 bb 60   24
50a8 : a5 49 48 a5 4a 48 20 fd   1c
50b0 : ae 20 8a ad 20 ea b9 a9   5a
50b8 : 10 a0 52 20 28 ba 68 a8   fc
50c0 : 68 aa 20 d4 bb 60 a5 49   08
50c8 : 48 a5 4a 48 20 fd ae 20   6b
50d0 : 8a ad a2 15 a0 52 20 d4   43
50d8 : bb 20 64 e2 a2 1a a0 52   3b
50e0 : 20 d4 bb a9 15 a0 52 20   6e
50e8 : a2 bb 20 6b e2 a9 1a a0   02
50f0 : 52 20 0f bb 68 a8 68 aa   50
50f8 : 20 d4 bb 60 a5 49 48 a5   8e
5100 : 4a 48 20 fd ae 20 8a ad   a8
5108 : 20 0e e3 a9 e0 a0 e2 20   3c
5110 : 50 b8 68 a8 68 aa 20 d4   f1
5118 : bb 60 a5 49 48 a5 4a 48   01
5120 : a9 00 8d 21 52 20 fd ae   cc
5128 : 20 8a ad a2 15 a0 52 20   2d
5130 : d4 bb 20 58 bc a9 bc a0   42
5138 : b9 20 5b bc f0 0a 2a b0   d9
5140 : 07 68 68 a2 0e 4c 37 a4   53
5148 : a9 15 a0 52 20 a2 bb a9   48
5150 : 15 a0 52 20 28 ba a9 bc   c6
5158 : a0 b9 20 50 b8 20 71 bf   b9
5160 : a9 15 a0 52 20 0f bb 20   b0
5168 : 0e e3 ad 21 52 d0 07 68   90
5170 : a8 68 aa 20 d4 bb 60 a5   f3
5178 : 49 48 a5 4a 48 a9 ff 8d   85
5180 : 21 52 20 fd ae 20 8a ad   04
5188 : 20 2b 51 a9 e0 a0 e2 20   a6
5190 : 50 b8 a9 00 8d 21 52 68   a3
5198 : a8 68 aa 20 d4 bb 60 a5   1b
51a0 : 49 48 a5 4a 48 20 fd ae   9b
51a8 : 20 8a ad a2 15 a0 52 20   ad
51b0 : d4 bb 20 fd ae 20 8a ad   9b
51b8 : 20 aa b1 8c ac 52 c8 8c   c5
51c0 : 21 52 18 ad da 51 69 05   ae
51c8 : 8d da 51 ad dc 51 69 00   ca
51d0 : 8d dc 51 20 fd ae 20 8a   0f
51d8 : ad a2 a8 a0 52 20 d4 bb   05
51e0 : ac 21 52 88 d0 d9 a9 a8   96
51e8 : 8d da 51 a9 52 8d dc 51   13
51f0 : a9 15 a0 52 20 a2 bb a9   f0
51f8 : ac a0 52 20 59 e0 68 a8   1d
5200 : 63 aa 20 d4 bb 60 7b 0e   29
5208 : fa 35 0f 86 65 2e e0 d2   22
5210 : 7f 5e 5b d8 aa 00 00 00   5b
5218 : 00 00 00 00 00 00 00 00   19
5220 : 00 00 e7 a7 0b 50 72 50   ad
5228 : 8d 50 a8 50 c6 50 fc 50   95
5230 : 1a 51 77 51 9f 51 00 00   7f
5238 : 00 00 00 00 00 00 00 00   39
5240 : 00 00 00 00 00 00 41 55   f0
5248 : 53 00 42 4f 47 00 47 52   4c
5250 : 44 00 44 4c 47 52 00 43   bd
5258 : 4f 54 00 41 43 4f 54 00   fa
5260 : 41 52 43 53 00 41 52 43   df
5268 : 43 00 50 4f 4c 59 00 00   39
5270 : 00 00 00 00 00 00 00 00   71
5278 : 00 00 00 00 00 00 00 00   79
5280 : 00 00 00 00 00 00 00 00   81
5288 : 00 00 00 00 00 00 00 00   89
5290 : 00 00 00 00 00 00 00 00   91
5298 : 00 00 00 00 00 00 00 00   99
52a0 : 00 00 00 00 00 00 00 00   a1
52a8 : 00 00 00 00 00 00 00 00   a9
52b0 : 00 00 00 00 00 00 00 00   b1
52b8 : 00 00 00 00 00 00 00 00   b9
52c0 : 00 00 00 00 00 00 00 00   c1
52c8 : 00 00 00 00 00 00 00 00   c9
52d0 : 00 00 00 00 00 00 00 00   d1
52d8 : 00 00 00 00 00 00 00 00   d9
52e0 : 00 00 00 00 00 00 00 00   e1
52e8 : 00 00 00 00 00 00 00 00   e9
52f0 : 00 00 00 00 00 00 00 00   f1
52f8 : 00 00 00 00 4a 63 63 63   0d
```

Listing 31. Die assemblierte MSE-Version von Listing 30

Anzahl der Speicherplätze im Stapelregister) sind hier ohne Bedeutung, weil ohnehin das Programm abgebrochen und der READY-Status aktiv wird. Die Fehlernummern und ihre Zuordnungen zeigt Ihnen die folgende Übersicht:

Fehler	Meldung	Fehler	Meldung
1	TOO MANY FILES	15	OVERFLOW
2	FILE OPEN	16	OUT OF MEMORY
3	FILE NOT OPEN	17	UNDEF'D STATEMENT
4	FILE NOT FOUND	18	BAD SUBSCRIPT
5	DEVICE NOT PRESENT	19	REDIM'D ARRAY
6	NOT INPUT FILE	20	DIVISION BY ZERO
7	NOT OUTPUT FILE	21	ILLEGAL DIRECT
8	MISSING FILENAME	22	TYPE MISMATCH
9	ILLEGAL DEVICE NUMBER	23	STRING TOO LONG
10	NEXT WITHOUT FOR	24	FILE DATA
11	SYNTAX	25	FORMULA TOO COMPLEX
12	RETURN WITHOUT GOSUB	26	CAN'T CONTINUE
13	OUT OF DATA	27	UNDEF'D FUNCTION
14	ILLEGAL QUANTITY		

Es gibt noch zwei weitere Meldungen: »28 VERIFY« und »29 LOAD«, die auf diese Weise aufgerufen werden können.

Kernel-Routinen

Die drei nächsten Adressen NEWSTT ($A7AE, 42926), GONE1 ($A7E4, 42980) und INTEND ($A7E7, 42983) sollen uns in diesem Zusammenhang noch nicht interessieren. Es handelt sich um verschiedene Stellen der Interpreterschleife, in die nach Bearbeitung einer eigenen Routine gesprungen wird oder die den normalen Inhalt eines Vektors bilden. Wir werden auf diese Interpreterschleife in einer späteren Folge noch ausführlich zurückkommen.

FRMNUM ist eine der wichtigsten Interpreter-Routinen, die zum Einlesen eines numerischen Wertes in den FAC dient:

FRMNUM

Holt beliebigen numerischen Ausdruck aus dem Basic-Text in den FAC und überprüft den Ausdruck.
- $AD8A dezimal 44426
- viele verschiedene Speicherstellen, darunter auch FAC
- benötigt alle Register
- Stapelbedarf verschieden, je nach Ausdruck

FRMNUM erledigt eine Vielzahl von nützlichen Aufgaben quasi nebenher: Die Speicherstelle $0D erhält den Inhalt 0, wenn ein numerischer Ausdruck, aber $FF, wenn ein Stringausdruck angesprochen wird. Im letzteren Fall erfolgt auch eine Fehlermeldung. In der Speicherstelle $0E wird angezeigt, ob man eine Fließkomma- oder eine Integerzahl geholt hat: Bei Fließkommazahlen findet man hier den Inhalt 0, bei Integer-Zahlen den Inhalt $80. Liegt eine einfache Variable vor, dann ist anschließend ein Zeiger in $45/$46 auf das erste Byte des Variablennamens gerichtet etc. Ich empfehle Ihnen, sich einmal ein ROM-Listing zu dieser Routine anzusehen. Sie werden noch einiges Brauchbares mehr entdecken.

CHKCOM

Prüft, ob das gerade gelesene Byte ein Komma ist und überliest dieses oder Ausgabe einer Fehlermeldung, wenn kein Komma vorliegt.
- $AEFD dezimal 44797
- Speicherstellen TXTPTR (also $7A und $7B)
- Register A und Y
- kein Stapelbedarf

Man verwendet diese Routine CHKCOM, um eine gewisse Struktur in die selbstgeschaffenen Basic-Befehle zu bringen. Es kann sonst unter Umständen leicht geschehen, daß der Interpreter den Basic-Text falsch liest.

FACINX

Wandelt eine Fließkommazahl im FAC in eine 2-Byte-Integer-Zahl um, die dann im Akku und Y-Register steht.
- $B1AA dezimal 45482
- Vorbereitungen: Zahl in FAC
- benötigt alle Register

Das entspricht der INT-Funktion des Basic, das LSB befindet sich im Akku, das MSB im Y-Register.

GETBYTC

Liest aus dem Basic-Text eine Zahl zwischen 0 und 255 ins X-Register ein.
- $B79B dezimal 47003
- Speicherstellen mehrere, unter anderem auch FAC
- benötigt alle Register

Diese Routine GETBYTC bringt gegenüber FRMNUM nur den Vorteil einer gewissen Bequemlichkeit. Im Grunde ruft sie nämlich zuerst einfach FRMNUM auf, dann FACINX und stellt noch eine Reihe von Überprüfungen an.

FSUB

Subtrahiert den FAC von einer durch Akku und Y-Register angezeigten Zahl im Speicher und legt das Ergebnis im FAC ab.
- $B850 dezimal 47184
- Vorbereitungen: A/Y als Vektor auf Zahl im Speicher richten, FAC laden
- Speicherstellen: mehrere, unter anderem auch FAC und ARG
- benötigt alle Register

FSUB lädt erst die durch A/Y angezeigte MFLPT-Zahl in den ARG, dreht dann das Vorzeichen des FAC-Inhaltes um und addiert beide.

Die nächste Adresse EINS ($B9BC, 47548) und auch die später auftretende PHIALB ($E2E0, 58080) ist jeweils die Startadresse einer Zahl, die schon fest im ROM im MFLPT-Format verankert vorliegt. Wie schon die Namen sagen, liegt bei EINS der MFLPT-Wert von 1 und bei PIHALB der von Pi/2. Solche Zahlenwerte gibt es einige im ROM.

...und Interpreter-Routinen

LOG

Bildet den natürlichen Logarithmus des FAC-Inhaltes und legt diesen dann im FAC ab.
- $B9EA dezimal 47594
- Vorbereitungen: Zahl in FAC
- Speicherstellen mehrere, unter anderem auch FAC und ARG
- benötigt alle Register

Diese Routine LOG prüft auch die Korrektheit des Argumentes im FAC. Die nächste Routine dient der Multiplikation zweier Fließkommazahlen:

FMULT

Der FAC-Inhalt wird mit einer MFLPT-Zahl multipliziert, auf die der Zeiger A/Y weist und das Ergebnis im FAC abgelegt.
- $BA28 dezimal 47656
- Vorbereitungen: Faktor 1 in FAC laden, Zeiger A/Y auf Faktor 2 richten
- Speicherstellen mehrere, unter anderen auch FAC und ARG
- benötigt alle Register

FDIV

Eine MFLPT-Zahl, auf die der Zeiger A/Y weist, wird

durch den Inhalt des FAC geteilt und das Ergebnis im FAC abgelegt.
- $BB0F dezimal 47887
- Vorbereitungen: Divisor in FAC, Zeiger A/Y auf Dividenden richten
- Speicherstellen mehrere, unter anderen auch FAC und ARG
- benötigt alle Register

Die Routine FDIV meldet auch einen Fehler, wenn der Dividend gleich Null ist. Zur Gruppe der Transportroutinen gehören die beiden folgenden:

MOVFM

Transportiert eine MFLPT-Zahl, auf die A/Y weist, aus dem Speicher in den FAC und wandelt sie dabei um in das FLPT-Format.
- $BBA2 dezimal 48034
- Vorbereitungen: Zeiger A/Y auf erstes Byte der Zahl im Speicher richten
- Speicherstellen $22, $23, FAC
- Register: A und Y

Den umgekehrten Weg öffnet diese Routine:

MOVMF

Transportiert den Inhalt des FAC in den Speicher an die Stelle, auf die X/Y weist und wandelt das FLPT- ins MFLPT-Format um.
- $BBD4 dezimal 48084
- Vorbereitungen: Zeiger X/Y auf erstes Byte im Speicher richten
- Speicherstellen $22, $23, FAC
- benötigt alle Register

Beide Routinen transportieren eigentlich nicht, sondern sie kopieren die entsprechenden Inhalte nur. Daraus folgt, daß die jeweilige Quelle unverändert erhalten bleibt.

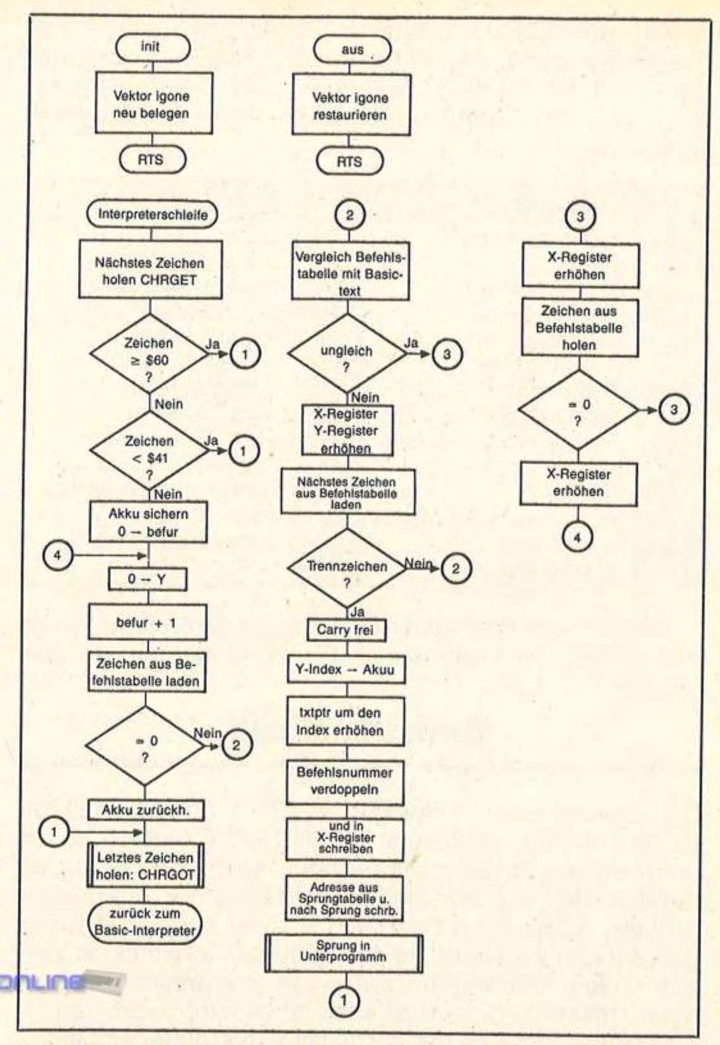


Bild 20. Das Flußdiagramm zum Modul 1 (erster Teil von Listing 30) der Befehlserweiterung

Mathematische Routinen

ABS

Ermittelt den Absolutwert einer Zahl im FAC
$BC58 dezimal 48216
- Vorbereitungen: Zahl in FAC
- Speicherstellen FAC
- keine Register

Ein sehr kurzes Programm, sehen Sie mal in ein ROM-Listing. Entspricht etwa der ABS-Funktion in Basic.

FCOMP

Vergleicht den FAC-Inhalt mit einer Zahl im Speicher auf die A/Y weist.
- $BC5B dezimal 48219
- Vorbereitungen: Zahl 1 in FAC, Zeiger A/Y auf Zahl 2 richten
- Speicherstellen $24, $25, FAC
- benötigt alle Register

Das Ergebnis des Vergleiches FCOMP wird im Akku angezeigt. Dabei ergeben sich folgende Aussagen:

Akku	Aussage
$01	FAC-Inhalt größer als Speicherzahl
$00	FAC-Inhalt gleich Speicherzahl
$FF	FAC-Inhalt kleiner als Speicherzahl

Nun kommen wieder einige Funktionen. Zunächst einmal das Bilden der Quadratwurzel des FAC-Inhaltes:

SQR

Die Quadratwurzel des FAC-Inhalts wird gebildet und im FAC abgelegt.
- $BF71 dezimal 49009
- Vorbereitungen: Zahl in FAC
- Speicherstellen mehrere, darunter auch FAC und ARG
- benötigt alle Register

POLYX

Berechnen eines Polynoms, Ergebnis im FAC.
- $E059 dezimal 57433
- Vorbereitungen: Argument in FAC, Zeiger A/Y auf Anfang einer Konstantentabelle, mehrere Speicherstellen,
- benötigt alle Register

POLYX wird von uns später beim POLY-Befehl verwendet. Die Konstantentabelle muß folgende Angaben enthalten:
1. Byte: Polynomgrad, weitere Bytes enthalten die Koeffizienten in absteigender Reihenfolge (also a_n, a_{n-1},...) als MFLPT-Zahlen.

COS

Bildet den Cosinus des FAC-Inhaltes und legt diesen im FAC ab.
- $E264 dezimal 57956
- Vorbereitungen: Zahl in FAC
- benötigt alle Register

Diese Funktion COS entspricht der Cosinus-Funktion im Basic ebenso wie die folgende Funktion SIN der Sinus-Funktion des Basic entspricht:

SIN

Bildet den Sinus des FAC-Inhalts und legt diesen im FAC ab.
- $E26B dezimal 57963
- Vorbereitungen: Zahl in FAC
- benötigt alle Register

Zu guter Letzt verwenden wir auch noch die Arcustangens-Funktion des Interpreters:

ATN
Bildet den Arcustangens des FAC-Inhaltes und legt diesen wieder im FAC ab.
- $E30E dezimal 58126
- Vorbereitungen: Zahl in FAC
- benötigt alle Register

Wir werden später bei den einzelnen Modulen direkt mit all diesen Interpreter-Routinen arbeiten.

Das Programm im Modul 1

Ab Zeile 480 (im Listing 30) fangen zwei kurze Programmteile an, die dem Ein- und Ausschalten der Befehlserweiterung dienen. Wie Sie sehen, speichern wir einfach in den Vektor IGONE die Startadresse unserer eigenen Interpreterschleife. Von da an wird jeder Befehl aus dem Basic-Text zuerst durch unsere Schleife und erst danach durch die normale Interpreterschleife überprüft. Das verlängert – allerdings unmerklich – die Abarbeitung eines Programms. Deshalb kann mit dem Programmstück ab Zeile 560 wieder der normale Inhalt des IGONE-Vektors restauriert werden. Wie Sie gleich sehen werden, passiert das immer dann, wenn AUS als neuer Basic-Befehl auftritt.

Ab Zeile 640 tritt nun unsere eigene Interpreterschleife auf den Plan. Unsere neuen Befehle sind als normale ASCII-Buchstaben im Basic-Text abgelegt. Deshalb wird ein durch die CHRGET-Routine in den Akku geholtes Zeichen zunächst überprüft, ob es sich um einen Buchstaben handelt. Falls das nicht der Fall ist, geben wir die Kontrolle wieder an den normalen Basic-Interpreter zurück. In Modul 10 haben wir eine Reihe von Tabellen und Hilfszellen geschaffen. Eine davon – nämlich AKKU – dient zur Zwischenspeicherung des Akku-Inhaltes, in eine andere – BEFNR – schreiben wir über das X-Register eine Befehlsnummer 0. Außerdem packen wir für die spätere Verwendung ins Y-Register den Wert 0 und erhöhen die Befehlsnummer auf 1. Des weiteren enthält das Modul 10 die Tabelle der Befehlstexte, aus der wir nun ein Zeichen in den Akku holen, überprüfen, ob wir ein Trennzeichen vor uns haben (nämlich eine Null). Nun wird verglichen (nämlich ab Zeile 850) und zwar Byte für Byte, ob der Text in der Befehlstabelle und der aus dem Basic-Text übereinstimmt. Sobald eine Ungleichheit festgestellt wird, überlesen wir schnell den Rest des Befehlswortes und überprüfen den nächsten Eintrag in der Tabelle. Mit jedem neuen Befehlswort wird auch der Inhalt in BEFNR erhöht.

Wenn der Basic-Text und der Text in der Befehlstabelle übereinstimmt, dann sorgen wir zuerst für das Erhöhen des TXTPTR, damit dieser Zeiger hinter unseren eigenen Befehl weist. Die Befehlsnummer in BEFNR wird verdoppelt und als Index in eine weitere Tabelle, die Sprungtabelle SPRTAB in Modul 10 verwendet. In dieser Tabelle liegen nacheinander die Startadressen aller Programmteile, die zu den einzelnen Befehlen gehören. Die Verdoppelung von BEFNR wird einfach dadurch nötig, daß jede Adresse aus zwei Byte besteht. In Zeile 1070 unseres Moduls steht nun ein Sprungbefehl, dessen Adresse noch aus einem Dummy-Wert besteht. Das LSB der Adresse ist sprung+1, das MSB sprung+2. Die aus der Sprungtabelle SPRTAB gelesene Adresse schreiben wir nun anstelle des Dummy-Wertes: Das Programm modifiziert sich an dieser entscheidenden Stelle selbst. Sobald das geschehen ist, sind wir schon bei dem Sprung in das Unterprogramm (also in ein anderes Modul) angekommen. Danach – also nach der Abarbeitung des Befehls – begeben wir uns zurück in den normalen Basic-Interpreter. In Bild 20 finden Sie noch ein Flußdiagramm, das Ihnen zum besseren Überblick über das Modul 1 dienen soll.

Die neuen Befehle

Interessant wird es nun bei den einzelnen neuen Befehlen: Wir lernen dabei die Anwendung der mathematischen Interpreter-Routinen kennen. Sie werden sehen, daß alles einfacher ist, als man gemeinhin denkt. Um welche Befehle geht es? Zur Erinnerung: Zunächst machen wir es uns etwas einfacher, indem wir Bogen- in Gradmaß durch BOG und GRD ausrechnen. Auch stört es viele, daß man immer mit der LOG-Funktion den reichlich ungebräuchlichen, natürlichen Logarithmus erhält: DLGR liefert uns den dekadischen Logarithmus. Der Vollständigkeit halber ist den Winkelfunktionen SIN, COS und TAN nun auch noch der COT (Cotangens) hinzugefügt. Die eingangs erwähnte Umkehrfunktion des Sinus (ARCS) ist ebenso vorhanden wie die des Cosinus (ARCC) und des Cotangens (ACOT). Mit der ohnehin schon vorhandenen ATN-Funktion für den Arcustangens ist somit auch der Satz der Umkehrfunktionen komplett. Zu guter Letzt gibt es da noch die POLY-Funktion, mit deren Hilfe man mit einem einzigen Befehl den Wert eines Polynoms berechnen kann.

Modulaufbau

Jedes Programm-Modul besteht aus einem Rahmen und einem Kern (siehe dazu Bild 21).

Der Rahmen ist überall nahezu identisch und soll daher nur einmal erklärt werden. Damit jeder neue Befehl sowohl im Programm- als auch im Direktmodus betrieben werden kann und die Ergebnisausgabe in Variablen erfolgt, hatten wir eine etwas ungewöhnliche, aber einfach zu durchschauende Technik gewählt: Das Ergebnis landet immer in der zuletzt aufgerufenen Variablen. So erfolgt der BOG-Aufruf beispielsweise durch

```
A = A:BOG,45
```

Das Ergebnis steht dann in A, was durch »PRINT A« leicht zu kontrollieren ist. Auf die Startadresse eines Variablenwertes weist der Zeiger »FORPNT«. Um sicherzugehen, daß er bei den manchmal reichlich unübersichtlichen Interpreter-Routinen nicht doch mal überschrieben wird, schieben wir seinen Inhalt auf den Stapel:

```
LDA FORPNT
PHA
LDA FORPNT+1
PHA
```

Es folgt der Aufruf einer Syntaxkontrolle:

```
JSR CHKCOM
```

CHKCOM überprüft, ob im Basic-Text ein Komma vorliegt. Ist das der Fall, wird es einfach überlesen. Andernfalls meldet sich der Computer mit SYNTAX ERROR und das Programm endet im READY-Zustand. Weshalb diese unnötige Kontrolle, werden Sie fragen! Zum einen wird ein Programm besser lesbar, wenn klar erkennbare Trennzeichen eingeplant sind. Das können durchaus auch andere als das Komma sein (mit der Komma-Abfrage geht's aber besonders leicht). Zum anderen werden Sie staunen, was ein Basic-Interpreter alles aus so einem selbstgemachten Befehl ohne Trennzeichen herauslesen kann!

Der letzte Befehl im oberen Teil unseres Rahmens ist

```
JSR FRMNUM
```

Damit lesen wir den Ausdruck hinter dem Komma in den Fließkomma-Akku 1 (den FAC) ein. Diese Routine nimmt nur numerische Ausdrücke an, bei Strings meldet sie einen Fehler und der Computer geht in den READY-Zustand. Die

damit eröffnete Variationsbreite der Möglichkeiten von numerischen Ausdrücken ist ungeheuer vielfältig: Wir können Integer-Zahlen, Fließkommaausdrücke und Festkommazahlen einlesen, einfache Variable oder Array-Elemente, kompliziert zusammengesetzte Formeln oder Funktionen. Das Ergebnis liegt danach immer in leicht verarbeitbarer Form im FAC vor.

Nach dieser Zeile folgt in allen Modulen der jeweilige Kern. Den Abschluß bildet danach der zweite Teil des Rahmens, der lediglich den zuvor auf dem Stapel gespeicherten Variablenzeiger zurückholt und den FAC-Inhalt in die dadurch bezeichnete Variable schreibt:

```
PLA
TAY
PLA
TAX
JSR MOVMF
RTS
```

Wie Sie sicher wissen, funktioniert der Stapelspeicher nach dem LIFO-Prinzip: Das heißt »Last In - First Out« und bedeutet, daß der zuletzt daraufgelegte Wert als erster wieder heruntergenommen wird (wie bei einem Bücherstapel). Als letzter Wert wurde im ersten Teil unseres Modulrahmens das MSB des Vektors FORPNT+1 abgelegt. Der kommt also nun wieder zurück in den Akku und von dort aus ins Y-Register. Das LSB lag darunter, wird als zweiter Wert vom Stapel geholt und auf dem Umweg über den Akku ins X-Register transportiert. Damit haben wir die Vorbereitungen schon erledigt, die die Interpreter-Routine MOVMF braucht: LSB der Zieladresse ins X-Register, MSB ins Y-Register. Mittels des MOVMF-Aufrufes transportieren (genaugenommen kopieren) wir nun den Inhalt des FAC in den dafür bereitgehaltenen Speicherbereich der durch FORPNT markierten Variable. Nun steht uns das Ergebnis vom Basic aus leicht zur Verfügung. Die Module werden durch RTS beendet, was den Rücksprung in das Modul 1, die Hauptschleife unserer Basic-Erweiterung, bewirkt.

Modul 2: BOG

Wir werden nun immer nur noch die Modulkerne besprechen. Da hatten wir es zuerst also mit dem Kern von BOG zu tun. BOG soll aus einem Winkel im Gradmaß das Bogenmaß berechnen. Nach dem Aktivieren der Erweiterung kann das Bogenmaß von 60 Grad im Direktmodus beispielsweise durch:

```
A=A:BOG,60:PRINT A
```

ermittelt werden. Unser Modul folgt der Umrechnungsformel:

$$\text{Bogenmaß} = \text{Gradmaß} * (\pi/180)$$

Der Faktor $\pi/180$ wurde BOGFAK genannt und sein Wert (0,0174532925) als Fließkommazahl in einer Konstantentabelle ab der BOGFAK genannten Speicherstelle abgelegt. Weil sich im FAC schon der Winkel im Gradmaß befindet, brauchen wir nur noch die Interpreter-Routine FMULT aufrufen, nachdem wir die Startadresse von BOGFAK in den Akku (LSB) und das Y-Register (MSB) geschrieben haben. Das Ergebnis der Multiplikation befindet sich im FAC. In Bild 21 finden Sie das Flußdiagramm.

Modul 3: GRD

Für die Umrechnung vom Bogen- in das Gradmaß gelten die gleichen Erläuterungen wie eben bei BOG: GRD leistet uns diesen Dienst nach der Gleichung:

$$\text{Gradmaß} = \text{Bogenmaß} * (180/\pi)$$

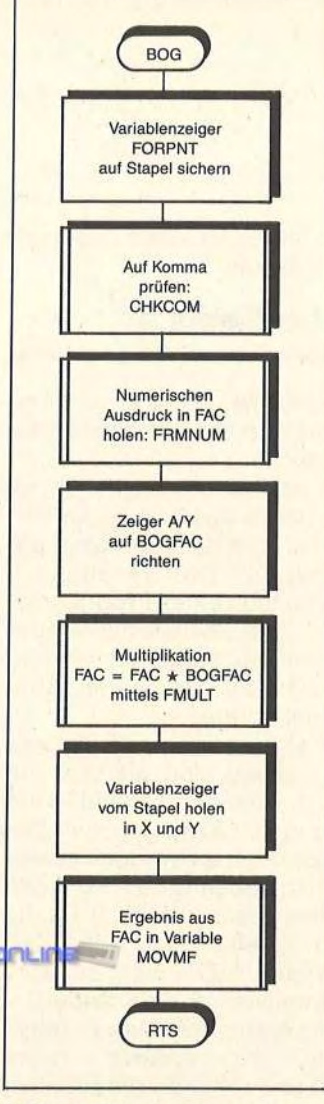


Bild 21. Wir erweitern den Basic-Wortschatz: allgemeine Struktur der Programm-Module am Beispiel der neuen Befehle BOG, GRD und DLGR

COT
Variablenzeiger auf Stapel sichern
Auf Komma prüfen: CHKCOM
Numerischen Ausdruck in FAC holen: FRMNUM
Wert sichern in ZWSP1 mittels MOVMF
Cosinus bilden: COS FAC = COS (FAC)
Ergebnis sichern in ZWSP2 mittels MOVMF
Anfangswert zurück aus ZWSP1 in FAC mittels MOVFM
Sinus bilden: SIN FAC = SIN (FAC)
Division ausführen FDIV FAC = ZWSP2/FAC
Variablenzeiger vom Stapel holen: X, Y
Ergebnis aus FAC in Variable: MOVMF
RTS

Bild 22. Das Flußdiagramm der Cotangens-Funktion

Auch diesen Faktor (er heißt GRDFAK und hat den Wert 57.2957795) finden Sie im Fließkommaformat in der Konstantentabelle ab GRDFAK. Außer dem anderen Faktor unterscheidet sich das Modul 3 nicht vom Modul 2.

Modul 4: DLGR

Spätestens an dieser Stelle werden Sie sich fragen, weshalb nicht der einprägsamere Befehlsname DLOG gewählt wurde. Da spielt uns der Interpreter wieder einen Streich: Beim Eintippen von DLOG und <RETURN> am Ende der Zeile würde er nämlich das LOG als normales Basic-Be-

fehlswort deuten und statt der ASCII-Zeichenkette LOG ein sogenanntes Token in den Programmtext einbauen, also eine Kennzahl, die der Computer beim Programmlauf später der Logarithmusfunktion zuordnet. Natürlich kann man als gewiefter Assembler-Programmierer auch diesem unerwünschten Interpreterverhalten einen Riegel vorschieben, aber dann wird es für diesen Kurs ein wenig unübersichtlich. Deshalb heben wir uns solche Feinheiten für eine spätere Folge auf und begnügen uns damit, Befehlsworte zu verwenden, die keine normalen Basic-Worte in sich enthalten.

Den dekadischen Logarithmus (log, also den Logarithmus zur Basis 10) berechnet man aus dem natürlichen (ln, also dem zur Basis e – dabei ist e die Eulersche Zahl 2,718281828459...) durch die Gleichung:

log (x) = ln (x) * (1/ln10)

Wieder haben wir einen Faktor vorliegen (nämlich 1/ln10, LOGFAK mit dem Wert 0,434294482), der als Fließkommazahl in der Konstantentabelle ab LOGFAK abgelegt ist. Diesmal wird aber nicht direkt der Wert im FAC mit der Konstanten multipliziert, sondern zuvor muß vom FAC-Inhalt noch der natürliche Logarithmus gebildet werden. Dazu verwenden wir – einfach durch den Routineaufruf mittels JSR – die Interpreterroutine LOG. Automatisch legt diese Routine den ln des eingegebenen Wertes im FAC ab, wo wir damit dann genauso weiterverfahren wie bisher: Akku und Y-Register auf LOGFAK richten und FMULT aufrufen.

Wie benutzt man DLGR? Hier ein Beispiel:

```
A=A:DLGR,Ausdruck:PRINT A
```

In A steht dann der dekadische Logarithmus von »Ausdruck«, der hier auf dem Bildschirm angezeigt wird.

Modul 5: COT

Bild 22 zeigt Ihnen zum besseren Verständnis das Flußdiagramm zur Cotangens-Funktion:

Wir berechnen den Cotangens nach der Formel:

cot(x) = cos(x) / sin(x).

Der im FAC gespeicherte Wert x muß also zweimal verarbeitet und die Ergebnisse dann miteinander durch Division verknüpft werden.

Dazu müssen wir den Wert x zunächst einmal zwischenspeichern. Zu diesem Zweck befinden sich direkt hinter der Tabelle mit den im letzten Teil erwähnten Fließkommakonstanten zwei weitere Zwischenspeicher, die »ZWSP1« und »ZWSP2« genannt wurden. Mittels der Routine MOVMF wird der FAC-Inhalt in den Zwischenspeicher ZWSP1 kopiert. Wie wir schon festgestellt haben, bleibt der FAC-Inhalt dabei erhalten, und wir wenden nun die COS-Funktion darauf an. Das Ergebnis befindet sich ebenfalls im FAC und wird durch den erneuten Aufruf von MOVMF in den anderen Zwischenspeicher ZWSP2 transportiert. Jetzt soll der Sinus des ursprünglichen Wertes gebildet werden. Dazu holen wir ihn mit dem Unterprogramm MOVFM aus dem ZWSP1 wieder in den FAC und benutzen die SIN-Routine. Was steht jetzt an welcher Stelle?

Im FAC steht SIN(X) und im ZWSP2 COS(X). Beide dividieren wir nun mit der Routine FDIV. Dazu brauchen wir nur einen Zeiger (MSB im Y-Register und LSB im Akku) auf den ZWSP2 richten und die Routine durch »JSR FDIV« aufzurufen. Das sollte man sich merken: Bei FDIV ist der FAC-Inhalt der Divisor und der durch den Zeiger A/Y gekennzeichnete Wert der Dividend:

FAC = (durch A/Y bezeichneter Wert) / FAC.

Das Ergebnis befindet sich dann wieder im FAC und ist der gesuchte Cotangens.

Modul 6: ACOT

Das Modul 6 berechnet die Umkehrfunktion des Cotangens nach der Formel:

acot(x) = (π/2) - atn(x).

Glücklicherweise ist der Fließkommaausdruck von π/2 schon im ROM unseres Computers fest verankert: er steht ab Adresse $E2E0. Diese Adresse haben wir in Modul 1 PIHALB genannt. Das Bild 23 zeigt Ihnen das Flußdiagramm des ACOT-Moduls:

Das im FAC eingetroffene Argument X wird sogleich mittels »JSR ATN« zum Arcustangens verarbeitet, der ebenfalls im FAC steht (um Mißverständnisse zu vermeiden: Der alte FAC-Inhalt X wird natürlich durch den neuen FAC-Inhalt ATN(X) überschrieben). Indem wir nun wieder einen Zeiger (LSB im Akku und MSB im Y-Register) einrichten, der auf PIHALB weist, und dann die Routine FSUB benutzen, bilden wir die Differenz. Auch hier sollte man sich merken: Bei FSUB steht der Subtrahend im FAC, und der Minuend wird durch den Zeiger markiert. Der Differenzwert erscheint im FAC:

FAC = (durch A/Y bezeichneter Wert) - FAC.

Dieser Differenzwert ist schon der gesuchte Arcuscotangens.

Modul 7: ARCS

Im Modul 7 geht es uns um die Umkehrfunktion des Sinus, den Arcussinus. Die Formel dafür ist etwas komplizierter als die bisher benutzten:

arcs(x) = atn(x/sqr(1-x^2)).

Zudem benötigen wir den Arcussinus auch noch später im Modul 8 zur Berechnung des Arcuscosinus. Zu dem Zweck ist im Rahmen des Moduls noch eine Speicherstelle namens FLAG berücksichtigt worden. In FLAG befindet sich der Wert 0, wenn das Modul nur den Arcussinus berechnet, aber der Wert $FF, wenn es über die Einsprungstelle EASIN vom Modul 8 aus aufgerufen wird. Das Flußdiagramm dieses Moduls finden Sie in Bild 24:

Der Modulkern speichert zunächst mit der uns schon bekannten Routine MOVMF das Argument im Zwischenspeicher 1. Falls Ihnen die weiteren Schritte suspekt vorkommen, hier die Erklärung: Sowohl der Arcussinus als auch der Arcuscosinus sind nur für Argumente x definiert, deren absoluter Wert (also deren Betrag, worunter man die Zahl x ohne Vorzeichen versteht) kleiner oder gleich 1 sind. Eine Eingabe von »ARCS,3« beispielsweise würde unweigerlich zu einem Fehler führen, denn wir erhielten eine Quadratwurzel über einem negativen Ausdruck, was eine komplexe Zahl als Argument der ATN-Funktion zur Folge hätte. Wir müssen daher vor der weiteren Verarbeitung überprüfen, ob /x/≦ 1 als Bedingung erfüllt ist.

Zu diesem Zweck bilden wir nach dem Zwischenspeichern den Absolutwert des eingegebenen Argumentes x, indem wir die ABS-Routine aufrufen. Danach befindet sich im FAC der Betrag von x. Wir vergleichen nun diesen FAC-Inhalt mit der Zahl 1. Der Fließkommawert von 1 findet sich gleich dreimal im ROM: Bei $B9BC, $BDE8 und $E376. Den bei $B9BC nennen wir EINS und richten einen Zeiger darauf, indem wir das LSB dieser Adresse in den Akku, das MSB ins Y-Register schreiben. Dann rufen wir die Routine FCOMP auf, die nun den FAC-Inhalt mit der Zahl vergleicht, auf die der Zeiger weist. Das Ergebnis des Vergleichs befindet sich im Akku: Er enthält 0, wenn beide gleich sind. Falls der FAC-Inhalt kleiner als die angezeigte Zahl ist, befindet sich im Akku $FF. Ist die angezeigte Zahl größer, steht im Akku eine 1.

Durch »BEQ ARGOK« verzweigen wir zur weiteren Berechnung, wenn Gleichheit festgestellt wurde, der Akku also Null enthielt. Ob $FF im Akku steht, stellen wir fest, indem wir mit ROL das Bit 7 des Akku ins Carry rotieren. Ist das Carry-Bit gesetzt, also der FAC-Inhalt kleiner als die ausgewiesene Speicherzahl, wird mit dem Befehl »BCS ARGOK« verzweigt. Ist aber nicht verzweigt worden, dann fiel das Argument x nicht in den vorgeschriebenen Rahmen. In diesem Fall holen wir den Stapelinhalt – den Zeiger FORPNT hatten wir dort ja abgelegt – und sorgen für die Ausgabe einer Fehlermeldung. Wie das geschieht und welche Meldungen zur Verfügung stehen, haben wir in der letzten Folge gesehen: Die Fehlernummer wird im X-Register abgelegt (hier entspricht $0E dem »ILLEGAL QUANTITY ERROR«) und das Programm durch »JMP ERROR« verlassen. Nach der Fehlermeldung geht der Computer in den READY-Status.

Sehen wir uns nun an, wie ab ARGOK (das kommt von »ARGument OK«) weiter verfahren wird: Zunächst holen wir das Argument wieder durch MOVFM aus dem Zwischenspeicher in den FAC. Auch bei diesem Transport handelt es sich in Wirklichkeit nur um ein Kopieren. Das machen wir

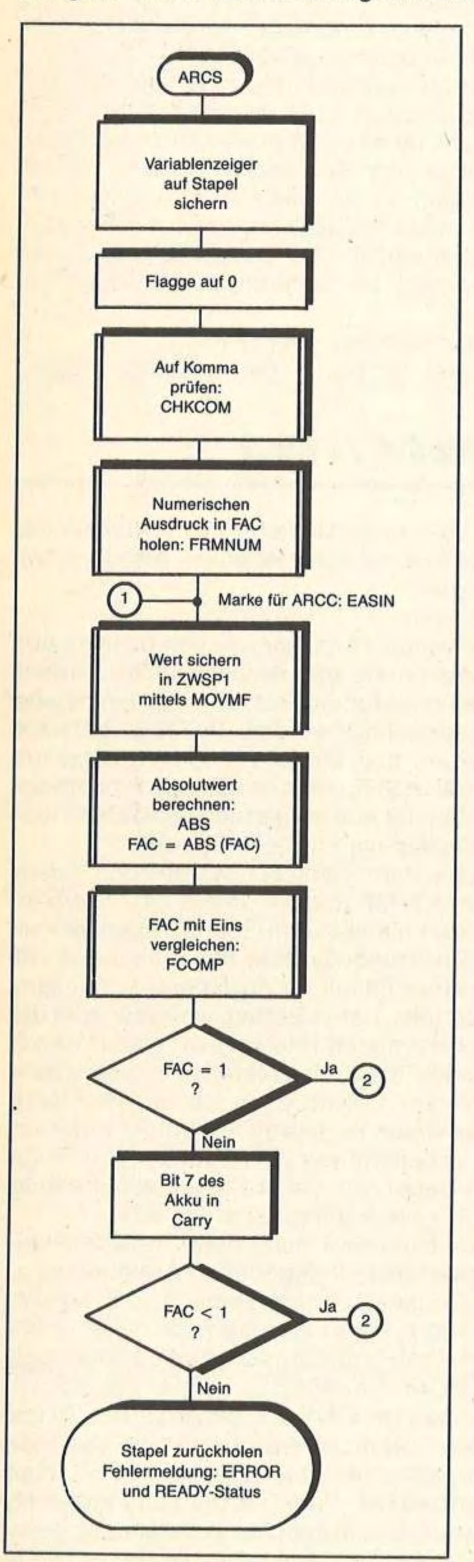


Bild 23. Mit dem Modul 6 kann das erweiterte Basic auch den Arcuscotangens berechnen

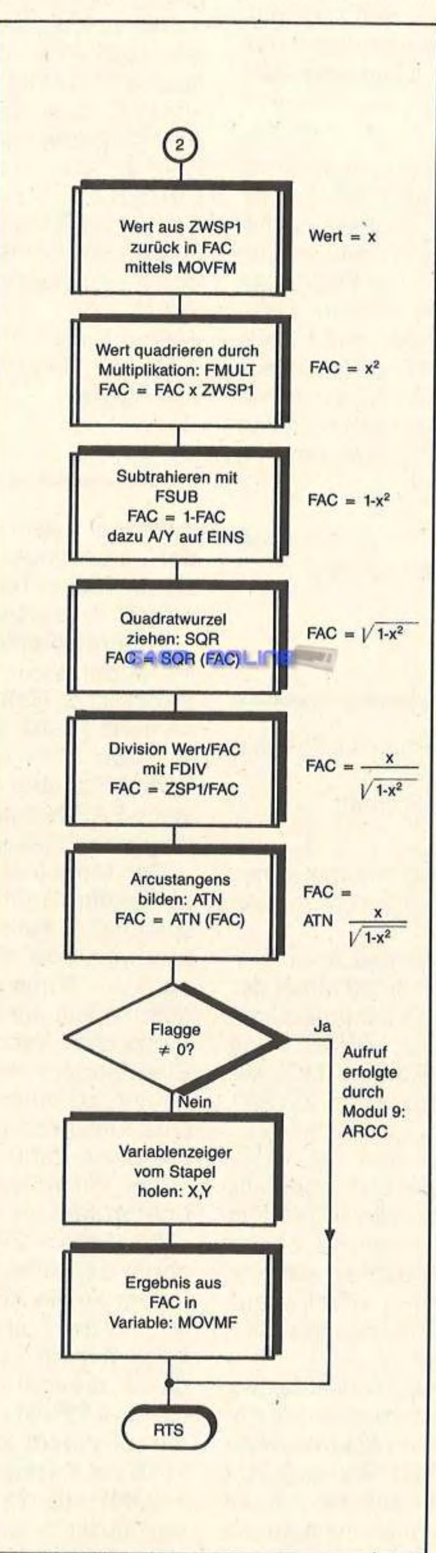


Bild 24. Das Modul 7 berechnet den Arcussinus. Dazu benutzen wir ausgiebig die Interpreterroutinen.

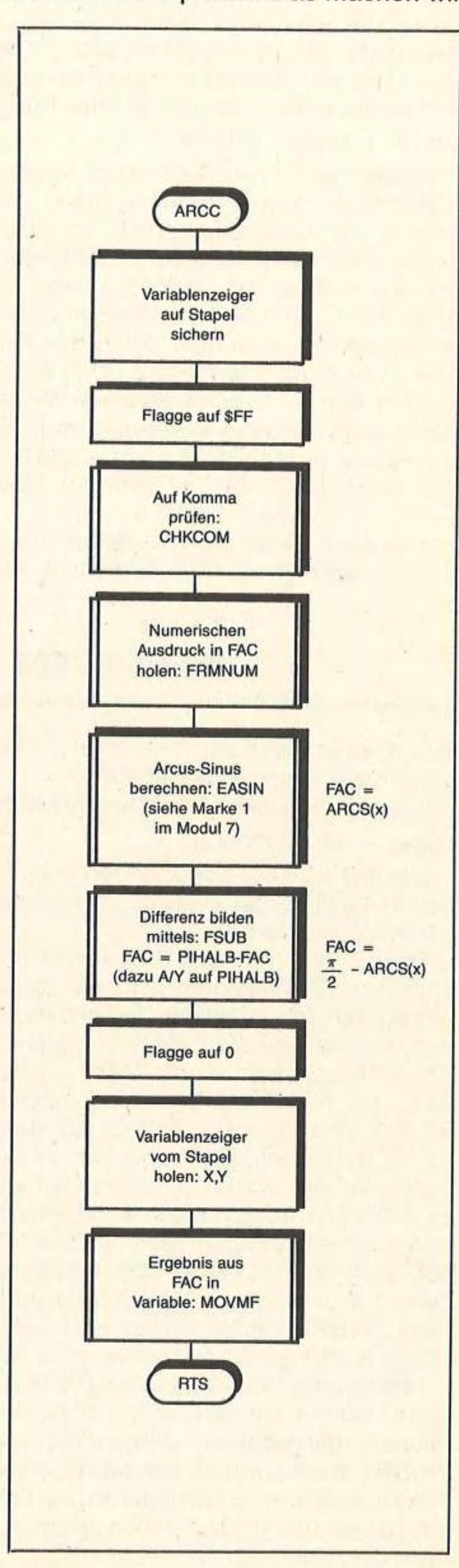


Bild 25. Den Arcuscosinus berechnet unser erweitertes Basic mit dem Modul 8

uns zunutze, indem wir nun sofort FMULT aufrufen, um den FAC-Inhalt (das ist ja x) mit dem Inhalt des Zwischenspeichers 1 (auch das ist x) zu multiplizieren. Danach steht x^2 im FAC. Erinnern Sie sich an FSUB und den ROM-Wert EINS? Wir können direkt danach den A/Y-Zeiger wieder auf EINS richten und FSUB aufrufen. Im FAC steht anschließend das Ergebnis der Funktion $1-x^2$. Die Quadratwurzel dieses Inhaltes bilden wir nun durch Aufruf der SQR-Routine: Im FAC ist dann $sqr(1-x^2)$ gespeichert. Nun richten wir noch einmal den A/Y-Zeiger auf den Zwischenspeicher 1 (dort befindet sich immer noch das Argument x), um FDIV aufzurufen. Sie erinnern sich: Der FAC ist der Divisor. Jetzt sind wir schon fast am Ziel, denn im FAC befindet sich nun schon $x/sqr(1-x^2)$. Wir benutzen noch die ATN-Routine »JSR ATN«, um nun im FAC den Arcussinus zu finden.

Den Abschluß des Kerns bildet noch die Prüfung der Speicherstelle FLAG. Wenn sich dort $FF befindet, kam der Aufruf des Moduls ja vom Arcuscosinus-Programm her, und wir müssen dorthin zurückspringen, ohne den Stapel zu leeren und das Ergebnis in die Variable zu schreiben. Durch »BNE RETOUR« überspringen wir diesen Teil des Modulrahmens, falls in FLAG ein Inhalt ungleich 0 steht.

Modul 8: ARCC

Durch die im Modul 7 geleistete Arbeit wird das Arcuscosinus-Modul recht einfach. In Bild 25 finden Sie das Flußdiagramm:

Den Arcuscosinus berechnen wir nach der Formel:

$arcc(x) = (\pi/2) - arcs(x)$. Bedingung dabei: $/x/ \leqq= 1$.

Im Rahmen des Moduls belegen wir zuerst die Speicherstelle FLAG mit dem Wert $FF, um in der Arcussinus-Routine eine Unterscheidung treffen zu können, wie der Aufruf erfolgte. Sofort nach dem Eintreffen des Argumentes im FAC steuern wir die Stelle EASIN im Modul 7 an. Nach der Rückkehr aus diesem Modul enthält der FAC den Arcussinus von x. Wir brauchen nur noch den Zeiger A/Y auf die vorhin schon benutzte ROM-Zahl PIHALB zu richten und FSUB aufzurufen, um schließlich den Arcuscosinus im FAC zu haben.

Modul 9: POLY

Im POLY-Modul kommt es uns nicht darauf an, Rechnungen durchzuführen. Vielmehr erschöpft sich die Arbeit in der richtigen Übernahme aller Parameter. Sehen wir uns zunächst einmal an, welche Werte die von uns benutzte Routine POLYX erwartet. Ein Polynom ist ein mathematischer Ausdruck der Form

$y = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_2 \cdot x^2 + a_1 \cdot x + a_0$

Einige Beispiele sollen das erläutern:

$y = 5 \cdot x^3 - 2 \cdot x + 7$ mit $n=3$, $a_3 =5$, $a_2 =0$, $a_1 =-2$ und $a_0 =7$

$y = 3.7 \cdot x^{12} - 5 \cdot x^4$ mit $n=12$, $a_{12} =3.7$, $a_4 =-5$,

alle anderen a-Werte (man nennt diese Werte Koeffizienten) sind gleich 0.

Sie sehen, es gibt unzählige Varianten. Außerdem können die Koeffizienten auch noch alle möglichen mathematischen Ausdrücke darstellen. POLYX erwartet vor dem Aufruf die Startadresse einer Tabelle im schon bekannten Zeiger A/Y. Der Aufbau dieser Tabelle sieht so aus:

1.Byte: Polynomgrad n (1-Byte-Integer)

Byte 2 bis 6: Koeffizient a_n (Fließkommazahl im MFLPT-Format)

Byte 7 bis 11: Koeffizient a_{n-1} (Fließkommazahl im MFLPT-Format)

Byte 12 bis 16: Koeffizient a_{n-2} (Fließkommazahl im MFLPT-Format)

und so weiter bis zum Koeffizienten a_0.

Der Aufruf soll in dieser Form erfolgen:

POLYX,x,n,a_n,a_{n-1},...,a_0

Im ersten oben genannten Beispiel stünde dann:

A=A:POLY,x,3,5,0,-2,7:
PRINT A

Unser Modulkern speichert also zunächst die Zahl x (das Argument) mit der schon bekannten Routine MOVMF aus dem FAC in den Zwischenspeicher 1. Nun holt sich das Programm nach erneutem Prüfen auf ein Komma wieder durch FRMNUM den Polynomgrad n in den FAC. Mit der Routine FACINX wandeln wir den Fließkommawert von n um in eine 2-Byte-Integer-Zahl, deren LSB im Y-Register und deren MSB im Akku landet. Uns reicht das LSB, denn Polynome mit einem Grad größer als 255 sind wohl kaum sinnvoll und lassen sich auch nicht mehr eingeben.

Die für POLYX reservierte Tabelle haben wir ans Ende des letzten Moduls gelegt und POLYTAB genannt. Den 1-Byte-Integerwert aus dem Y-Register schreiben wir durch »STY POLYTAB« dort hinein. Diesen Wert verwenden wir in der folgenden Einleseschleife auch als Zähler, denn wir haben genau n+1 Koeffizienten in die Tabelle zu schreiben. Mit INY erhöhen wir also den Zähler um 1 und legen ihn in der Speicherstelle FLAG ab: Durch die nachfolgenden Interpreter-Routinen wird nämlich das Y-Register verändert.

Im ersten Modul hatten wir eine Speicherstelle POLYVAR definiert, die POLYTAB-4 entsprach. Diese Speicherstelle spielt in der folgenden Einleseschleife eine wichtige Rolle. Sie ist die Zieladresse, die über das X- und das Y-Register an die Transportroutine MOVMF gegeben wird. Die beiden entsprechenden Zeilen im Programm sind durch »M1« und »M2« markiert. Am Anfang der Schleife (nach der Marke »m0«) addieren wir zu POLYVAR (und zwar dem Wert, der in den Speicherstellen M1+1 und M2+1 steht) in einer 16-Bit-Addition die Zahl 5: Jede MFLPT-Zahl nimmt 5 Byte für sich in Anspruch. Das Ergebnis dieser Addition (Selbstmodifikation!) wandert zurück in die Speicherstellen M1+1 und M2+1. Nach dieser Addition lesen wir den jeweils nächsten Koeffizienten in den FAC und transferieren ihn mit MOVMF an die jeweils durch den neuen POLYVAR-Wert ausgewiesenen Tabellenplatz. Danach laden wir wieder FLAG, dekrementieren diesen Zähler um 1 und prüfen mit BNE, ob noch weitere Koeffizienten zu laden sind.

Sind alle Koeffizienten durch diese Schleife in der Tabelle gelandet, greifen wir nochmals zur Selbstmodifikation, indem wir in M1+1 und M2+1 wieder den Originalwert von POLYVAR eintragen. Die Tabelle ist nun komplett. Wir holen das Argument x durch MOVFM wieder aus dem Zwischenspeicher 1 in den FAC, richten nun – wie vorhin besprochen – den Zeiger A/Y auf die Tabelle und rufen die Routine POLYX auf. Im FAC befindet sich das Ergebnis, das wir in unseren Erklärungen mit y bezeichnet haben. Ein Flußdiagramm dieses Moduls finden Sie in Bild 26.

Tabellenmodul

Noch ein paar Worte sollen zum Tabellenmodul gesagt werden. Sowohl in der Sprung- als auch in der Befehlstext-Tabelle ist noch Platz für etwa acht weitere neue Basic-Befehle. Die Polynomtabelle ab POLYTAB ist auf den Polynomgrad 16 vorbereitet. Selten dürften mehr Koeffizienten nötig sein. Falls aber doch: Diese Tabelle liegt am Programmende (und sollte dort auch nach Programmerweiterungen liegen), wodurch sich ohne Probleme noch beliebig viele Koeffizienten anschließen lassen.

Mit diesen recht ausführlichen Erklärungen des bisher umfangreichsten Beispielprogrammes sollten Sie nun in der Lage sein, auch selbst Basic-Befehlserweiterungen zu schreiben und die wichtigsten Interpreter-Routinen mathematischer Aufgabenstellungen zu verwenden. Sie haben es sicherlich bemerkt: Die Fließkommazahlen und ihre unterschiedlichen Formate im Computer spielen bei allen diesen Routinen eine wichtige Rolle. Bei nahezu allen Rechenoperationen (auch bei Verwendung von Integer-Zahlen wie A%) arbeitet der Interpreter mit Fließkommazahlen. Im nächsten Abschnitt werden wir uns diesen Zahlen und ihrer Darstellung zuwenden.

Erinnern Sie sich? Am Anfang dieses Kurses hatten wir allerlei Zahlensysteme zum Thema: Die Binärzahlen, die Hexadezimalzahlen und die uns geläufigen Dezimalzahlen. Wir waren aber so bequem, uns nur den ganzen Zahlen zu widmen, und wie Sie alle wissen, bilden die eher die Ausnahme als die Regel. Umgehen wir all die mathematischen Feinheiten und nennen die anderen die »Kommazahlen«, dann wissen Sie, wovon nun die Rede sein wird.

Fließkommazahlen

Sehen wir uns das Ganze erst einmal im gewohnten System der Dezimalzahlen an. 2,71 Meter lang war der größte Mensch der Welt (Robert Pershing Wadlow, 1919-1940), dessen Maße medizinisch nachgeprüft wurden. 8,5 Einwohner leben im Staat Vanatuatu (Ozeanien) auf einem Quadratkilometer – in der Bundesrepublik sind es 245,5. Der Atomkern eines Heliumatoms wiegt etwa 0,000 000 000 000 000 000 000 006 643 kg. Füllt man dieses Gas in einen Luftballon, dann befinden sich (bei normalen Druck- und Temperaturverhältnissen) etwa 26 900 000 000 000 000 000 Heliumatome in einem Kubikzentimeter. Die beiden letzten Beispiele veranlassen meist zum mehrfachen Nachprüfen der vielen Nullen. Weil es solche unhandlichen Zahlenungetüme öfters gibt, hat man sich eine etwas bequemere Darstellung der Zahlen einfallen lassen. Man nennt das dann manchmal die »wissenschaftliche Darstellung« oder einfach »Fließkommadarstellung« (ab und zu ist auch von »Gleitkommadarstellung« die Rede). Was versteht man darunter, und wie baut man solche unhandlichen Zahlen wie die eben genannten in Fließkommazahlen um?

Dazu geht man von der Basis unseres Zahlensystems – nämlich der Zahl 10 – und ihren Potenzen aus, also 10, 10x10, 10x10x10 oder auch 1/10 und so fort. Die etwas unbequeme Schreibweise der Zehnerpotenzen kann durch die Verwendung von Hochzahlen vereinfacht werden:

$10 \times 10 = 100 = 10^2$

$10 \times 10 \times 10 = 1000 = 10^3$

$10 = 10^1$

$1/10 = 0{,}1 = 10^{-1}$

Jede Zahl kann als Produkt mit einer solchen Zehnerpotenz dargestellt wer-

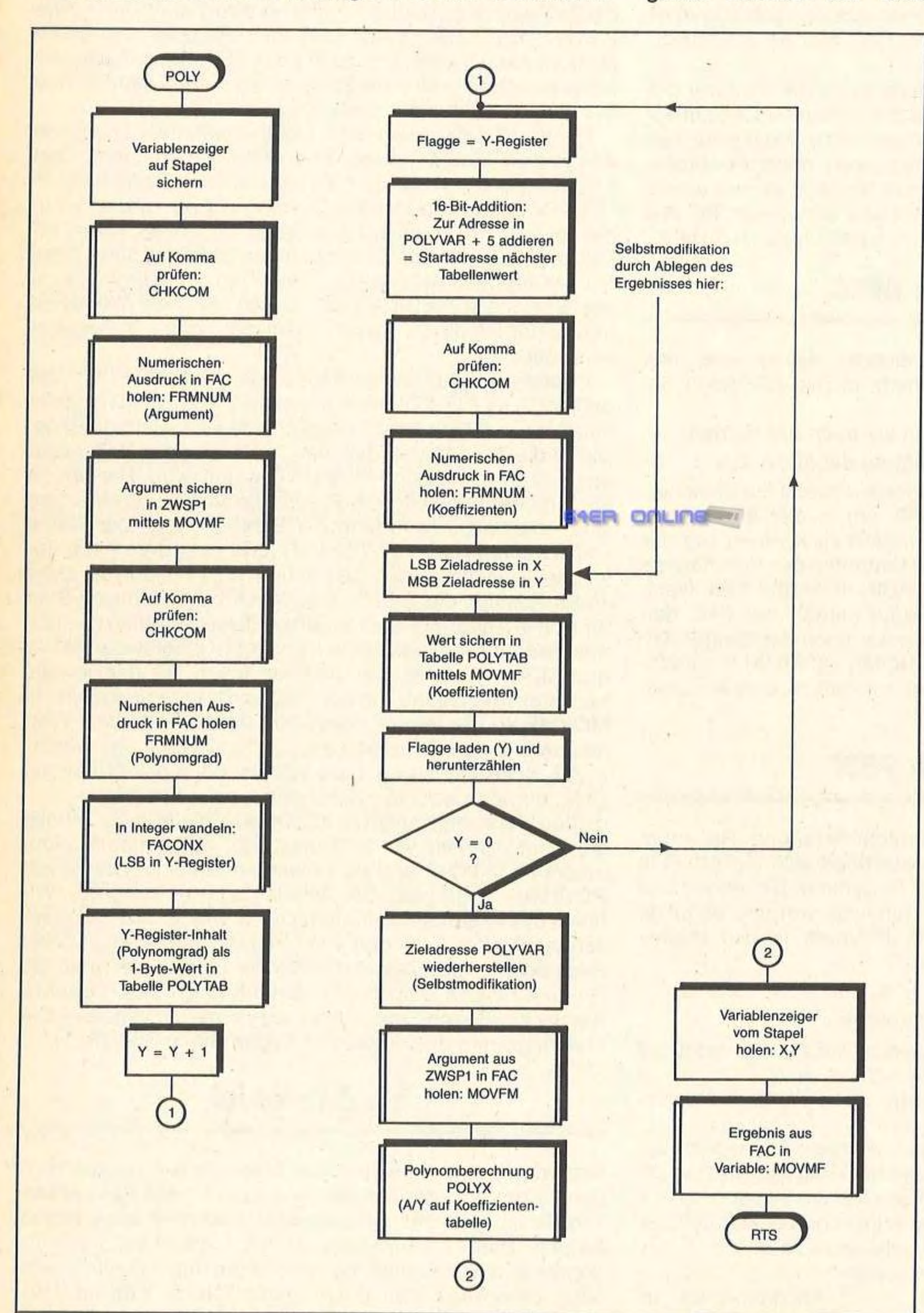


Bild 26. Das Flußdiagramm des neunten Moduls: Der Befehl POLY kann die Arbeit mehrerer Programmzeilen in Windeseile mit einem einzigen Befehl erledigen

den, ja es gibt genau genommen unendlich viele Möglichkeiten der Schreibweisen als Produkte. Ein Beispiel soll das zeigen: Nehmen wir die Zahl 1985,125. Diese kann man – rechnen Sie nach – auch wie in Bild 27a schreiben. Auch anders herum ist das natürlich möglich, wie Sie in Bild 27b ebenfalls sehen können. Alle dort in der letzten Spalte gezeigten Zahlen sind Fließkommadarstellungen derselben Zahl 1985,125. Vielleicht erkennen Sie nun auch, weshalb »Fließkomma«: Das Komma hat keinen festen Platz mehr. In Abhängigkeit von der Zehnerpotenz wechselt es seinen Ort. Als Faustregel kann man sich merken:

Jede Verschiebung des Kommas um eine Stelle nach links führt zur Erhöhung der Hochzahl um 1, jede Kommaverschiebung nach rechts aber zu einer Erniedrigung der Hochzahl um 1.

Wozu das Ganze? Entscheiden Sie selbst an den beiden vorhin genannten Zahlenungetümen: $6{,}643 \times 10^{-27}$ kg wiegt der Heliumatomkern und in einem Kubikzentimeter dieses Gases befinden sich $2{,}69 \times 10^{19}$ Atome. Man kann jede Zahl in einem festen und überschaubaren Format darstellen. Wo aber ist das entscheidender als im Speicher unseres Computers?

Die Zweifingerlinge und Fließkommazahlen

Für alles nun Folgende sollten Sie die Binärzahlen, ihren Aufbau und die Umrechnungsmethoden schon kennengelernt haben. Dies gilt besonders für den nächsten Abschnitt. Dieser hier kann recht kurz bleiben, denn natürlich ist es uns klar, daß auch die Zweifingerlinge nicht immer nur mit ganzen Zahlen zu tun haben. Außerdem treten bei ihnen noch viel eher als bei uns lange Zahlenungetüme auf, die durch eine Fließkommadarstellung lesbar gemacht werden müssen. Erinnern Sie sich: Die Zahl »4« beispielsweise benötigt in unserem Dezimalsystem nur eine Ziffer, im Binärsystem aber schon drei, nämlich %100 (übrigens werde ich im folgenden immer den Vorsatz »%« verwenden, wenn die dahinter stehende Zahl eine Binärzahl sein soll).

Wenn sich also im Land der Zweifingerlinge vier Personen (%100) eine Sachertorte teilen müssen, dann erhält jeder von ihnen %1/100 davon (als Bruch dargestellt) oder %0,01 (als Kommazahl dargestellt, also 0,25). Wieso %0,01? Es gibt im Prinzip zwei Wege der Erklärung. Sehen wir uns zunächst kurz den unbequemen Weg an, um nachher ausführlich den leichteren und breiter anwendbaren zu wählen.

Die Basis des Binärsystems ist ja die Zahl »2« (das ist %10). Denken Sie an die vorhin erwähnte Methode der Fließkommazahlen, dann sehen Sie sofort (naja, oder jedenfalls bald), daß:

%0,01 = $\%0{,}1 \times 10^{-1}$
oder = $\%1 \times 10^{-10}$ ist.

In Dezimalzahlen ausgedrückt ist $\%1 \times 10^{-10}$ aber gleich 1×2^{-2}, was dasselbe ist wie $1/(2^2)$, also ¼ oder 0,25. Aber nun schnell zum leichteren Weg!

Wir werden uns zu dieser Umrechnung aus dem dezimalen System in das der Zweifingerlinge zunächst einmal wieder unser Beispiel 1985,125 vornehmen und daran Schritt für Schritt erkennen, wie man vorgeht.

Dazu trennen wir den Umrechnungsvorgang in zwei Teile auf: Der Vorkomma-Anteil (also 1985) wird nämlich anders umgerechnet als der Nachkomma-Anteil (also 125). Wie das mit dem Teil vor dem Komma zu geschehen hat, ist bereits erklärt worden. Wir teilen die Zahl durch 2, notieren den Rest, teilen das Ergebnis wieder durch 2, notieren den Rest und so fort, bis wir irgendwann auf das Ergebnis 0 stoßen. Die gemerkten Reste aber bilden dann die gesuchte Binärzahl:

a)								
1985,125	=	198,5125	x	10	=	198,5125	x	10^1
oder	=	19,85125	x	100	=	19,85125	x	10^2
oder	=	1,985125	x	1000	=	1,985125	x	10^3
oder	=	0,1985125	x	10000	=	0,1985125	x	10^4
b)								
1985,125	=	19851,25	/	10	=	19851,25	x	10^{-1}
oder	=	198512,5	/	100	=	198512,5	x	10^{-2}
oder	=	1985125	/	1000	=	1985125,0	x	10^{-3}
oder	=	19851250	/	10000	=	19851250,0	x	10^{-4}

Bild 27a und 27b. Beispielzahl in verschiedener Schreibweise

1985 : 2 = 992	Rest 1
992 : 2 = 496	Rest 0
496 : 2 = 248	Rest 0
248 : 2 = 124	Rest 0
124 : 2 = 62	Rest 0
62 : 2 = 31	Rest 0
31 : 2 = 15	Rest 1
15 : 2 = 7	Rest 1
7 : 2 = 3	Rest 1
3 : 2 = 1	Rest 1
1 : 2 = 0	Rest 1

Von unten nach oben gelesen ergeben die Reste dann %111 1100 0001.

Übrig bleibt also der Nachkomma-Anteil, beziehungsweise die Zahl 0,125. Anstelle der Kettendivision durch 2 verwendet man hier nun die Kettenmultiplikation. Die umzuwandelnde Zahl wird mit 2 malgenommen, ein sich ergebender Vorkomma-Anteil notiert, dann die Nachkommastellen des Ergebnisses wieder mal 2 genommen, wieder Vorkomma-Anteil notiert und so fort. Das geschieht so lange, bis der Nachkomma-Anteil des Ergebnisses gleich 0 geworden ist. Sehen wir uns das an einigen Beispielen an. Wir stellen uns die Aufgabe, die Dezimalzahl 0,1 in die ihr entsprechende Binärzahl umzuwandeln:

0,1 x 2 = 0,2 Vorkommast. 0
0,2 x 2 = 0,4 Vorkommast. 0
0,4 x 2 = 0,8 Vorkommast. 0
0,8 x 2 = 1,6 Vorkommast. 1

Weiterrechnen ohne die neue Vorkommastelle:

0,6 x 2 = 1,2 Vorkommast. 1

Nochmal weiter ohne neue Vorkommastelle:

0,2 x 2 = 0,4 Vorkommast. 0
0,4 x 2 = 0,8 Vorkommast. 0

Sie dürfen gern weiterüben. Zu einem Ende werden Sie bei dieser Zahl nie gelangen, denn manchmal ergibt sich aus einem endlichen Dezimalbruch ein unendlicher (hier periodischer) Binärbruch. In der Reihenfolge der auftretenden Vorkommastellen angeordnet, erhält man die Nachkommastellen der Binärzahl, im Beispiel also:

%0,000 1100 1100 1100 . . .

Erinnern Sie sich noch an den 0,25-Anteil der Sachertorte? Dies als weiteres Übungsstück:

0,25 x 2 = 0,5 Vorkommast. 0
0,5 x 2 = 1,0 Vorkommast. 1

Der Nachkomma-Anteil ist 0 geworden und daher die Umrechnung beendet.

Als Ergebnis erhalten wir somit %0,01, was zu erwarten war. Nun können wir auch unser Beispiel 1985,125 weiter umrechnen:

0,125 x 2 = 0,25 Vorkommast. 0
0,25 x 2 = 0,5 Vorkommast. 0
0,5 x 2 = 1,0 Vorkommast. 1

Auch hier ist nun der Nachkomma-Anteil 0 geworden, die Rechnung daher beendet und das Ergebnis lautet %0,001.

Jetzt können wir beide Ergebnisse kombinieren zur kompletten Kommazahl der Zweifingerlinge:
1985,125 entspricht
%111 1100 0001,001

Fließkommazahlen im Computer

Fassen wir das Ganze nochmal kurz zusammen: Eine dezimale Fließkommazahl wird in ihr binäres Pendant umgerechnet durch Trennen des Vor- und des Nachkomma-Anteils. Der Vorkomma-Anteil wird dann durch Kettendivision, der Nachkomma-Anteil durch Kettenmultiplikation in die Binärzahl umgerechnet und beide wieder kombiniert zur Gesamtzahl.

Wie Sie wissen, ist der Computer ja in Wirklichkeit ein Gerät aus der Welt der Zweifingerlinge. Kommazahlen kennt er also auch nur als Binärzahlen. Außerdem hatten wir vorhin festgestellt, daß es dem Computer besser liegt, die Kommazahlen als Fließkommazahlen zu speichern, weil da das Format so schön einheitlich ist. Als Assembler-Programmierer ist man manchmal in der Verlegenheit, Fließkomma-Konstanten in einer Tabelle zur späteren Verwendung durch ein Programm abzulegen. Einige Beispiele konnten Sie im Beispielprogramm (Listing 30) finden, welches dem Basic mathematische Routinen hinzugefügt hat, die manche Anwender schmerzlich vermissen (es dreht sich um LOGFAK, BOGFAK und GRDFAK im Tabellenmodul ab Zeile 4010). Wie erwartet der Computer solche Werte und wie kann man sie in die gewünschte Form bringen? An zwei Beispielen werden wir uns das nun ansehen: Als erstes arbeiten wir mit der Zahl 1985,125 weiter, danach vollziehen wir den ganzen Weg einmal am Beispiel von GRDFAK.

Normalisieren

Der Computer bewahrt also Kommazahlen als binäre Fließkommawerte auf. Genauso, wie wir vorhin im Dezimalsystem an unserem Beispiel 1985,125 das Komma verschoben und damit die Hochzahl verändert haben, geschieht das jetzt im nächsten Prozeß, dem sogenannten »Normalisieren«. Darunter versteht man eine Verschiebung des Kommas so weit nach links (oder bei anderen Zahlen nach rechts), bis vor dem Komma nur noch eine Null steht, dahinter dann die erste von Null verschiedene Ziffer. Bei 1985,125 im Dezimalsystem führt das dann zu: 0,1985125 x 10^4.

Wir haben das Komma um 4 Stellen nach links geschoben, der Exponent (das ist ein anderer Name für »Hochzahl«) ist daher von 0 auf 4 angewachsen. Sehen wir uns nun das gleiche bei den Binärzahlen an. Aus
%111 1100 0001,001
wird dann:
%0,1111 1000 0010 01 x 10^{1011}

Bitte denken Sie daran, daß in diesem Fall %10 gemeint ist, also dezimal 2. Der Binärwert %1011 entspricht der Dezimalzahl 11 und um genau diese Anzahl Stellen ist das Komma nach links gewandert.

Der Exponent

Unsere Zahl ist nun eindeutig festgelegt durch die sogenannte Mantisse (worunter man die Zahl versteht, die zwischen dem Komma und dem »x«-Zeichen steht) und den Exponenten (solange die Basis unverändert %10 bleibt). Der Computer muß nun also zwei Größen festhalten (Mantisse und Exponent).

Möchten Sie die Beispiele am Computer nachvollziehen, müssen Sie statt eines Kommas einen Dezimalpunkt einsetzen.

Bis hierher galt das Gesagte unabhängig von jedem Computertyp. Jetzt aber unterscheiden sich die weiteren Vorgehensweisen – und zwar abhängig von der jeweiligen Interpreterstruktur. In allen mir bekannten 8-Bit-Commodore-Computern und einigen anderen, deren Interpreter auf der Microsoft-Basis arbeiten, wird aber ebenso verfahren, wie es hier nun erklärt wird.

Zur Speicherung des Exponenten ist ein Byte vorgesehen. Nun kann aber dieser Exponent (wie in unserem Beispiel) positiv oder auch negativ sein. Hier könnte man nun von der Praxis Gebrauch machen, das Bit 7 dieses Exponetenbytes als Vorzeichenbit zu verwenden (so geschieht das im Fall der Speicherung von Integers). Hier aber verwendet man noch ein anderes Verfahren. Zum Exponenten wird die Zahl 128 addiert! Das führt dann bei positiven Exponenten zu Werten, die größer als 128, bei negativen zu solchen, die kleiner als 128 sind.

So kann man auch leicht ermitteln, welches denn die größte und die kleinste Fließkommazahl sein kann, die unser Computer verarbeitet. Die Summe 128+Exponent darf nicht größer als 255 werden (das ist ja die größte in einem Byte speicherbare Zahl). Also kann der Exponent maximal 127 betragen. 2^{127} aber entspricht 1,7014118 x 10^{38}. Anders herum: Kleiner als 0 kann das Exponenten-Byte nicht werden –128 ist daher der kleinste mögliche Exponent und 2^{-128} entspricht der Zahl
2,9387358 x 10^{-39}.

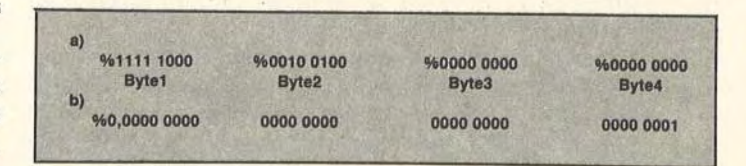
Kommen wir nun wieder zu unserem Beispiel 1985,125. Die Addition des berechneten Exponenten %1011 mit 128 (das ist %1000 000) ergibt den Inhalt des Exponenten-Bytes:
%1000 1011

Rechnen wir diesen Wert noch ins Dezimalsystem um, dann ergibt sich die Zahl 139 und im Hexadezimalsystem haben wir $8B.

Die Mantisse

Weil der Computer immer mit Binärzahlen arbeitet, die Basis 2 also vorausgesetzt wird, fehlt uns nun zur eindeutigen Festlegung der Fließkommazahl nur noch die Mantisse. Diese wird in 4 Byte gespeichert, und zwar linksbündig. Falls nur ein Teil der 4 Byte für die Ziffern benötigt wird, füllt unser Computer den Rest mit Nullen auf. In Bild 28a sehen Sie die Mantisse der Zahl 1985,125. In Dezimalzahlen entspricht das der Zahlenfolge 248, 36, 0, 0 und in Hexadezimalzahlen der Reihe $F8, $24, $00, $00.

Wie genau ist eigentlich unsere Fließkommadarstellung? Schon bei der Umrechnung der Dezimalzahl 0,1 ins System der Zweifingerlinge haben Sie bemerkt, daß wir auf unendlich viele Stellen weiterrechnen können, ohne den genauen Wert zu erhalten. Aber auch andere Dezimalzahlen füllen nach der Umrechnung manchmal mehr als nur die 4 Byte der Mantisse. Zweifellos sind die ersten 32 Bit der Mantisse die am meisten signifikanten. Ein Beispiel aus dem gewohnten Dezimalsystem soll das zeigen: Es ist ein großer Unterschied zwischen den Zahlen 0,1 und 0,9, ein



	Byte1	Byte2	Byte3	Byte4
a)	%1111 1000	%0010 0100	%0000 0000	%0000 0000
b)	%0,0000 0000	0000 0000	0000 0000	0000 0001

Bild 28 a) Mantisse von 1985, 125. b) Kleinster, mit vier Mantissen-Byte darstellbarer Unterschied.

kleiner aber nur zwischen 0,1000000001 und 0,1000000009. Immerhin, manchmal zählt auch dieser kleine Unterschied! Alle Bits ab Bit 32 (die also nicht mehr in die 4 Byte passen) fallen unter den Tisch. Der kleinste Unterschied zwischen zwei Zahlen, der in diesen vier Mantissen-Byte festgehalten werden kann, beträgt 2^{-32} oder in dezimal 0,000 000 000 2 (die binäre Schreibweise sehen Sie in Bild 28b). Daraus folgt, daß man beispielsweise die Zahlen 1,000 000 000 2 und 1,000 000 000 0 darstellen kann, nicht aber 1,000 000 000 1.

In der 10. Stelle rechnet unser Computer wegen seiner Mantissendarstellung in 4 Byte also ungenau, und je komplexer eine Rechnung wird, desto mehr breiten sich diese Ungenauigkeiten in die 9. oder sogar 8. Stelle aus. Das nennt man dann den Rundungsfehler.

Die Formate: FLPT und MFLPT

Eigentlich wäre jetzt schon alles geklärt, wenn wir nicht den Umstand vergessen hätten, daß auch die Mantisse ein Vorzeichen hat. Bisher konnten wir beispielsweise eine Zahl -1985,125 noch nicht darstellen. Auch dieses Problem ist natürlich gelöst, und zwar gleich auf zweierlei Weise. Es gibt nämlich zwei Formate, in denen Fließkommazahlen in unserem Computer stehen.

Das einfachere davon nennt man FLPT-Format (das kommt von »FLoating-PoinT«, was »Fließpunkt« bedeutet). Das Vorzeichen der Mantisse wird hier einfach in einem eigenen Byte gelagert. Das Bit 7 dieses Bytes ist 0, wenn wir eine positive, oder 1, wenn wir eine negative Zahl vor uns haben. Die restlichen 7 Bit (also Bit 6 bis 0) dieses Vorzeichen-Byte sind unbenutzt. Sehen wir uns nun in Bild 29a unser Beispiel 1985,125 im kompletten FLPT-Format an. Diese Lagerung und Verarbeitung einer Fließkommazahl in 6 Byte findet vor allem an zwei markanten Orten unseres Computers statt: Dem FAC und dem ARG. Beides sind sogenannte Fließkomma-Akkumulatoren, von denen der FAC (»Floating point ACcumulator«) für Fließkommazahlen etwa die Rolle des Akku spielt, also gewissermaßen die Rechenzentrale darstellt. Eine große Anzahl von Interpreterfunktionen erfordert das Argument im FAC und liefert das Ergebnis dorthin. Die USR-Funktion des Basic packt das Argument ebenfalls in den FAC, was eine bequeme Übergabemöglichkeit von Fließkommawerten an ein Maschinenprogramm darstellt. Dazu werden wir ein anderes Mal noch kommen.

Viele Interpreterfunktionen erfordern zwei Argumente. Eines davon liegt dann im FAC, das zweite im ARG (von »ARGument«, manchmal auch FAC2 genannt). Bild 30 zeigt Ihnen den Aufbau und den Ort des FAC und des ARG im C 64 und im C 128.

Fließkommazahlen werden aber nicht nur an diesen zwei Orten des Computers verwendet, meist müssen sie irgendwo im RAM ihr Dasein fristen als Variable, als Array-Elemente und so weiter. Da wäre es schon eine Speicherverschwendung, jedesmal 6 Byte für solche Werte zu reservieren, von denen eines nur für das Vorzeichen (als Bit 7) benötigt wird! Aus diesem Grund existiert noch ein gepacktes Format, das man MFLPT-Format nennt (das kommt von »Memory FLoating PoinT«). Hier findet die Fließkommadarstellung in nur 5 Byte statt. Wie kann man das erreichen, wo doch schon der Exponent und die Mantisse volle 5 Byte erfordern?

Ein Bit braucht man nur für das Vorzeichen. Gibt es in diesen 5 Byte ein überflüssiges Bit, das man dazu verwenden kann? Es gibt! Denken Sie an den Vorgang des Normalisierens, wo die Verschiebung des Kommas so weit gefordert wurde, daß vor dem Komma eine 0, danach aber die erste

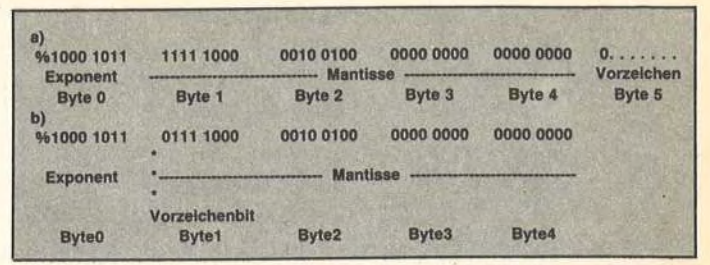


Bild 29 a) 1985,125 im FLPT- und b) im MFLPT-Format

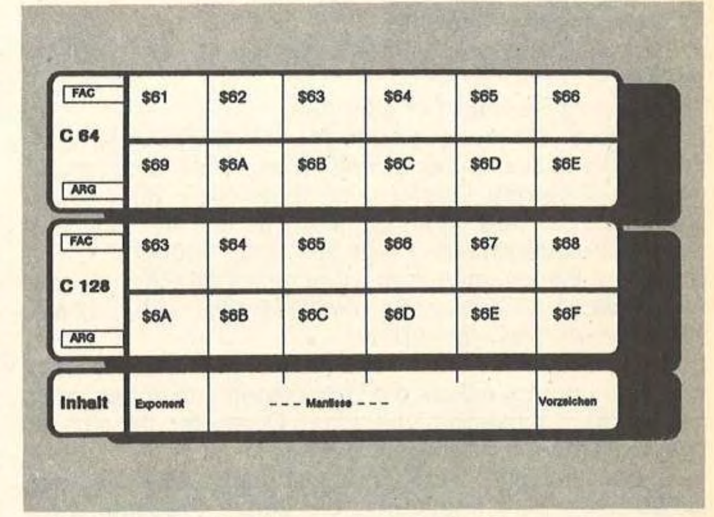


Bild 30. Orte und Aufbau der beiden Fließkomma-Akkumulatoren im C 64 und im C 128

signifikante Ziffer steht. Im Binärsystem gibt es aber für diese erste Stelle nach dem Komma nur eine Möglichkeit: Es muß sich um die Ziffer 1 handeln! Wenn das aber ohnehin klar ist, dann kann man auf dieses erste Mantissen-Bit auch verzichten. Behält man einfach immer im Sinn, daß dort auf alle Fälle noch eine 1 hingehört, dann kann man nun mit diesem Bit anstellen, was man möchte: es beispielsweise als Vorzeichenbit verwenden. Genau das geschieht, und deshalb finden Sie im Bit 7 des ersten Mantissen-Bytes immer eine 0, wenn wir eine positive Zahl, aber eine 1, wenn wir eine negative Zahl vor uns haben. In Bild 29b sehen Sie unser Beispiel 1985,125 im MFLPT-Format. Hier ergibt sich also die dezimale Zahlenfolge 139, 120, 36, 0, 0 oder im Hexadezimalsystem: $8B, $78, $24, $00, $00.

Eine komplette Umrechnung

Nun werden wir am Beispiel von Grdfak die komplette Berechnung durchführen: GRDFAK = 180/π = 180/3,141592... = 57,2957795...

a) Vorkomma-Anteil umrechnen:

57:2 = 28 Rest 1
28:2 = 14 Rest 0
14:2 = 7 Rest 0
7:2 = 3 Rest 1
3:2 = 1 Rest 1
1:2 = 0 Rest 1

Damit folgt für den Vorkomma-Anteil: %111001

b) Nachkomma-Anteil umrechnen:

0,2957795 x 2 = 0,591559 Vorkommaziffer 0
0,591559 x 2 = 1,183118 Vorkommaziffer 1
0,183118 x 2 = 0,366236 Vorkommaziffer 0

Rechnen Sie weiter, bis Sie mit dem eben ermittelten Vorkommateil 32 Stellen ermittelt haben. Sie erhalten dann die Kommazahl in Bild 31a.

a)
%11 1001,0100 1011 1011 1000 0011 0100 10
b)
%0,1110 0101 0010 1110 1110 0000 1101 0010 x 10^{110}
c)

%0110 0101	0010 1110	1110 0000	1101 0010
Byte 1	Byte 2	Byte 3	Byte 4

Bild 31 a) Errechneter Nachkommaanteil. b) Normalisieren c) GRDFAK im MFLPT-Format.

c) Normalisieren: Bild 31b
d) Exponent: Addieren von 128 ergibt nun für das Exponentenbyte:
%0110 oder dezimal 134 oder $86.
e) Mantisse: Wir brauchen den Wert GRDFAK im MFLPT-Format und lassen (er ist ja positiv) das erste Mantissen-Bit daher zu 0 werden. Das Resultat sehen Sie in Bild 31c. Es entspricht der dezimalen Zahlenfolge 101, 46, 224, 210 und der hexadezimalen Folge $65, $2E, $E0, $D2.
f) Eintrag: Insgesamt haben wir nun für GRDFAK in unseren Quelltext die Zahlenfolge $86, $65, $2E, $E0, $D2 einzuschreiben. Das wär's dann!

Ganz schön viel Arbeit, werden Sie sagen. Stimmt! Es ist natürlich unumgänglich, die Grundlagen zu kennen, aber Sie verfügen schließlich über einen Computer, der sich für solche Aufgaben besonders gut eignet.

Speziell bei einer häufigen Anwendung dieser Methode ist es wesentlich bequemer, die ganze Prozedur nicht immer selbst durchführen zu müssen. Sie sollten deshalb einmal versuchen, Ihrem Computer die Arbeit zu überlassen. Im nächsten Abschnitt dieses Kurses finden Sie eine mögliche Programmlösung dazu – für den C 64 und den C128.

Beide Programme (Listing 32 und 33) verwenden den sogenannten programmierten Direktmodus und steuern damit einen Maschinensprachemonitor an (für den C 64 muß man vor dem Start noch den SMON geladen haben!). Dabei läuft das C 128-Programm automatisch, beim C 64-Programm ist es noch nötig, nach der Monitormeldung viermal <RETURN> zu drücken (SMON scheint den Tastaturpuffer nicht in gewohnter Weise zu behandeln). Auf dem Bildschirm erscheint dann die Einschaltmeldung des Monitors. Nach Druck auf <RETURN> sehen Sie die Speicherbereiche ab $6000 und $6010. In diese Bereiche transportierte ein kleines Maschinenprogramm die zuvor eingegebene Zahl als MFLPT- (ab $6000) und als FLPT-Zahl (ab $6010). Das Maschinenprogramm findet sich in den DATA-Zeilen des Listings und ist dabei in REM-Zeilen als Quelltext dargestellt. Der Sprung in die Routine $BBD4 (genannt MOVMF) ist C 64-Benutzern schon aus den letzten Folgen vertraut: Die Register X und Y weisen als Zeiger auf eine Speicherstelle, in die durch MOVMF der Inhalt des FAC unter gleichzeitiger Umwandlung ins MFLPT-Format transportiert wird. C 128-Benutzer finden diese Routine ab Adresse $8C03. Ihre Funktionsweise unterscheidet sich nicht von der entsprechenden C64-Routine.

Fließkommazahlen per USR übergeben

Wie haben wir die Zahlen übrigens in den FAC hineinbekommen? Da gibt es das – vom Basic-Programmierer gemiedene – Kommando USR(n), wobei »n« ein beliebiges Argument sein kann. Dieses n nun findet man nach dem USR-Kommando im FAC vor. Wie funktioniert USR? Stößt der Interpreter auf dieses Kommando, dann führt er einen Sprung in ein Maschinenprogramm aus, dessen Adresse

```
10 REM ************************************
   *************                          <054>
20 REM *
               *                          <069>
30 REM *   PROGRAMM ZUM UMWANDELN VON ZAHL
   EN IN DIE   *                          <155>
40 REM *   C 64-FORMATE   MFLPT  (AB $6000
   )           *                          <059>
50 REM *                  FLPT   (AB $6010
   )           *                          <010>
60 REM *
               *                          <109>
70 REM *       HEIMO PONNATH` HAMBURG  198
   6           *                          <199>
80 REM *
               *                          <129>
90 REM ************************************
   *************                          <134>
100 REM                                   <162>
110 PRINT CHR$(147)"IST DER SMON AB $C000
    SCHON EINGELADEN{2SPACE}(J/N)";:INPUT
    A$                                    <043>
120 IF A$<>"J" THEN PRINT"WUENSCHE GUTEN A
    BSTURZ...ODER SMON LADEN!":END        <020>
130 FOR I=0 TO 17:REM EINLESEN DES MASCHIN
    ENPROGRAMMES                          <001>
140 READ D:POKE 828+I,D                   <223>
150 NEXT I                                <234>
160 REM -------- DAS MASCHINENPROGRAMM ---
    ---------                             <116>
170 DATA 162,000     :REM LDX #$00      ;LSB
     ZIELADRESSE                          <111>
180 DATA 160,096     :REM LDY #$60      ;MSB
       -"-                                <028>
190 DATA 032,212,187:REM JSR $BBD4      ;FAC
     -> (X/Y)                             <161>
200 DATA 162,006     :REM LDX #$06      ;ZAE
    HLER EINRICHTEN                       <165>
210 DATA 181,096     :REM LDA $60,X     ;FAC
     AUSLESEN                             <048>
220 DATA 157,015,096:REM STA $600F,X    ;UND
     UEBERTRAGEN                          <156>
230 DATA 202         :REM DEX           ;ZAE
    HLER -1                               <008>
240 DATA 208,248     :REM BNE $033D     ;WEI
    TER BIS FAC UEBERTRAGEN IST           <236>
250 DATA 096         :REM RTS           ;ZUR
    UECK INS BASICPROGRAMM                <048>
260 REM                                   <068>
270 REM ,-------- USR-VEKTOR AUF $828 RICHT
    EN -----                              <139>
280 REM                                   <088>
290 POKE 785,60 :REM LSB DES USR-VEKTORS  <065>
300 POKE 786,3  :REM MSB DESSELBEN        <061>
310 REM                                   <118>
320 REM -------- EINGABEN UND USR-AUFRUF -
    ---------                             <145>
330 REM                                   <138>
340 PRINT CHR$(147):INPUT"ZAHL EINGEBEN";A <014>
350 B=USR(A):REM B IST NUR EIN DUMMY      <189>
360 REM                                   <168>
370 REM ----- PROGR.DIREKTMODUS : MONITORA
    UFRUF -----                           <212>
380 PRINT CHR$(147)CHR$(17)               <235>
390 PRINT"SYS49152"CHR$(17)CHR$(17)CHR$(17
    )CHR$(17)                             <088>
400 PRINT" M 6000 6001"CHR$(17):REM HIER L
    IEGT DIE ZAHL IM MFLPT-FORMAT         <146>
410 PRINT" M 6010 6011"CHR$(17):REM UND HI
    ER IM FLPT-FORMAT                     <213>
420 PRINT" X"CHR$(17)                     <046>
430 PRINT"RUN490"                         <196>
440 PRINT CHR$(19);                       <035>
450 POKE 631,13                           <002>
460 POKE 198,1:END                        <045>
470 REM -----------------------------------
    -----------                           <195>
480 REM                                   <032>
490 PRINT:PRINT"AB $6000 MFLPT-FORMAT"    <076>
500 PRINT"AB $6010 FLPT-FORMAT"           <167>
510 PRINT:INPUT"WEITERE ZAHLEN (J/N)";A$  <179>
520 IF A$="J" THEN 340                    <106>
530 POKE 785,72:POKE 786,178:REM USR-VEKTO
    R AUF NORMALWERT                      <037>
540 END                                   <034>
```

Listing 32. Berechnung von FLPT- und MFLPT-Format für den C 64

```
10 REM ***************************************
   ***********
20 REM *
          *
30 REM *   PROGRAMM ZUM UMWANDELN VON ZAHLEN
   IN DIE  *
40 REM *   C128-FORMATE   MFLPT   (AB $6000)
          *
50 REM *                  FLPT   (AB $6010)
          *
60 REM *
          *
70 REM *      HEIMO PONNATH  HAMBURG  1986
          *
80 REM *
          *
90 REM ***************************************
   ***********
100 REM
110 FOR I=0 TO 17: REM EINLESEN DES MASCHINE
    NPROGRAMMES
120 READ D$: POKE DEC("1600")+I,DEC(D$)
130 NEXT I
140 REM -------- DAS MASCHINENPROGRAMM -----
    -------
150 DATA A2,00    : REM LDX #$00      ;LSB ZIE
    LADRESSE
160 DATA A0,60    : REM LDY #$60      ;MSB
    -"-
170 DATA 20,03,8C: REM JSR $8C03     ;FAC ->
    (X/Y)
180 DATA A2,06    : REM LDX #$06      ;ZAEHLER
     EINRICHTEN
190 DATA B5,62    : REM LDA $62,X     ;FAC AUS
    LESEN
200 DATA 9D,0F,60: REM STA $600F,X   ;UND UEB
    ERTRAGEN
210 DATA CA       : REM DEX           ;ZAEHLER
     -1
220 DATA D0,F8    : REM BNE $1609     ;WEITER
    BIS FAC UEBERTRAGEN IST
230 DATA 60       : REM RTS           ;ZURUECK
     INS BASICPROGRAMM
240 REM
250 REM -------- USR-VEKTOR AUF $1600 RICHTE
    N -----
260 REM
270 POKE DEC("1219"),0 : REM LSB DES USR-VEK
    TORS
280 POKE DEC("121A"),22: REM MSB DESSELBEN
290 BANK 15: REM SICHERHEITSHALBER
300 REM
310 REM -------- EINGABEN UND USR-AUFRUF ---
    -------
320 REM
330 PRINT CHR$(147): INPUT "ZAHL EINGEBEN";A
340 B=USR(A): REM B IST NUR EIN DUMMY
350 REM
360 REM ----- PROGR.DIREKTMODUS : MONITORAUF
    RUF -----
370 PRINT CHR$(147) CHR$(17)
380 PRINT "MONITOR" CHR$(17) CHR$(17) CHR$(1
    7) CHR$(17)
390 PRINT "M 06000 06001" CHR$(17): REM HIER
     LIEGT DIE ZAHL IM MFLPT-FORMAT
400 PRINT "M 06010 06011" CHR$(17): REM UND
    HIER IM FLPT-FORMAT
410 PRINT "X" CHR$(17)
420 PRINT "RUN480"
430 PRINT CHR$(19);
440 BANK 0: POKE 842,13: POKE 843,13: POKE 8
    44,13: POKE 845,13: POKE 846,13
450 POKE 208,5: END
460 REM ------------------------------------
    ----------
470 REM
480 PRINT : PRINT "AB $6000 MFLPT-FORMAT"
490 PRINT "AB $6010 FLPT-FORMAT"
500 PRINT : INPUT "WEITERE ZAHLEN (J/N)";A$
510 IF A$="J" THEN 330
520 POKE DEC("1219"),40: POKE DEC("121A"),12
    5: REM USR-VEKTOR AUF NORMALWERT
530 END
```

Listing 33. Und dasselbe für den C 128

Startadresse($)	Format	Inhalt
AEA8	MFLPT	Pi
B1A5	MFLPT	-32768
B9BC	MFLPT	1
B9C2	MFLPT	Polynomkoeffizienten für LOG-Berechnung
B9D6	MFLPT	SQR(1/2)
B9DB	MFLPT	SQR(2)
B9E0	MFLPT	-0.5
B9E5	MFLPT	ln 2
BAF9	MFLPT	10
BDB3	MFLPT	99 999 999.9
BDB8	MFLPT	999 999 999
BDBD	MFLPT	1 000 000 000
BF11	MFLPT	0.5
BFBF	MFLPT	1/ln2
BFC5	MFLPT	Polynomkoeffizienten für EXP-Berechnung
BFE3	MFLPT	ln 2
BFE8	MFLPT	1
E2E0	MFLPT	Pi/2
E2E5	MFLPT	2*Pi
E2EA	MFLPT	0.25
E2F0	MFLPT	Konstanten für die Entwicklung von SIN,COS,...
E309	MFLPT	2*Pi
E33F	MFLPT	Konstanten für die Entwicklung von ATN
E376	MFLPT	1

Tabelle 3. Die wichtigsten Zahlentabellen im ROM des C 64

als Vektor beim C 64 in den Speicherzellen $311/$312 (dezimal 785/786) gespeichert ist. Er weist im allgemeinen auf die Adresse $B248, wo die Ausgabe der Fehlermeldung »SYNTAX ERROR« ausgegeben und ein Programmabbruch ausgeführt wird. Der C 128 versteckt diesen Vektor in den Speicherstellen $1219/$121A (dezimal 4633/4634). Sein Inhalt zeigt normalerweise auf die Adresse $7D28, die den »ILLEGAL QUANTITY ERROR« behandelt.

In unseren beiden Programmen verbiegen wir einfach diese USR-Vektoren, so daß sie auf $1600 (C 128) oder $334 (C 64) zeigen, wohin wir unsere kleine Assembler-Routine gelegt haben. Der USR-Aufruf schaltet in dieses kleine Programm und transportiert das Argument n in den FAC. Wir könnten durch das M-Kommando des Monitors auch direkt in den FAC hineinsehen, würden dort aber nicht mehr unsere Zahl entdecken. Der FAC wird vom Zeitpunkt des USR-Aufrufes bis zur Ausführung des M-Kommandos verändert. Deshalb die Verschiebung des FAC-Inhaltes nach $6010.

Das USR-Kommando ist zweifellos die bequemste Methode, Fließkommazahlen von Basic aus an ein Assemblerprogramm zu übergeben. Leider ist das aber nur für einen Wert einfach. Werden es mehrere, dann steigt der Programmaufwand. Eine andere Methode haben wir in den letzten Folgen kennengelernt, nämlich die Übergabewerte durch FRMNUM aus dem Basic-Text zu lesen. Eine weitere Methode lernen wir in der kommenden Folge kennen: Variable werden vom Basic-Interpreter in einer Tabelle abgelegt, die man durchaus auch von Assemblerprogrammen her benutzen kann. Bevor wir uns aber diesen Möglichkeiten zuwenden, werden wir diesmal noch etwas mehr über Tabellen erfahren.

Zur Ausrüstung von Schülern und Studenten (und vielen anderen) gehörte früher auch ein ständig mitgeschlepptes Tabellenwerk, in dem sich dann beispielsweise die Logarithmen der Zahlen von 1 bis 1000 fanden oder die Sinuswerte der Winkel von 0 bis 90 Grad und vieles andere mehr. Dann kam die Revolution durch die Taschenrechner: Kein mühseliges Nachschlagen mehr, kein Interpolieren, hohe

Genauigkeit! Der Computer hat die Tabellen verdrängt ... oder doch nicht?

Zwar werden solche Arbeiten wie das Berechnen eines Sinus oder von Logarithmen im Computer durch Entwicklung von Potenzreihen erledigt. Das dauert aber verhältnismäßig lange und für besonders zeitkritische Programme greift der Assembler-Programmierer auf Tabellen zurück! Wir finden Tabellen in unserem Computer in verschiedenen Erscheinungsformen: als Zahlentabellen mit Integer- oder Fließkommawerten, als Texttabellen, als Adressentabellen und als Sprungtabellen.

Tabellen im ROM

Falls Sie mal in der Situation sein sollten, beispielsweise den Wert 2*Pi in einem Programm benutzen zu müssen, dann können Sie sich viel Rechnerei ersparen, mit der Sie diese Zahl in das MFLPT-Format bringen: Im ROM befindet sich 2*Pi nämlich schon abrufbereit, genauso wie eine ganze Reihe weiterer Zahlen und Texte. Die ROM-Bereiche unseres Computers liefern uns also nicht nur Assembler-Routinen, die wir ansteuern, sie sind auch eine Datenquelle. Damit Sie wissen, wo Sie was im Computer finden können, sehen Sie sich die hier abgedruckten Werte der Tabellen an.

Tabelle 3 listet die wichtigsten Zahlentabellen im ROM des C 64 auf. Die Tabelle 4 zeigt die Zahlentabellen des C128. Die Tabellen 5 und 6 beziehen sich auf die Texttabellen im ROM des C 64 und des C 128.

Einige weitere interessante Tabellen im ROM des C 128 listet Tabelle 7 auf. Schließlich finden Sie in Tabelle 8 noch die Sprungtabelle im C 128 und ihre Zuordnungen.

Außer den hier vorgestellten Tabellen finden sich natürlich noch weitere in den ROM-Bausteinen: Da gibt es Tabellen zur Decodierung der Tastatur, Tabellen von Farbwerten, Tabellen zur Initialisierung des Systems, die Default-Werte (Einschaltwerte) enthalten und so weiter.

Interessanter als die eben behandelten ROM-Tabellen sind natürlich Tabellen in eigenen Programmen. Nehmen wir einmal an, Sie benötigen in einem Programm sehr häufig irgendwelche Potenzen von 2 (also 2 hoch 3, 2 hoch 4 und so weiter). Die dabei vorkommenden Hochzahlen bewegen sich zwischen 0 und 7. Nun können Sie natürlich jedesmal den Potenzwert ausrechnen, beispielsweise bei der Zahl 2 hoch 5:

```
LDA #$02 ;Basis in den Akku laden, also 2
ASL      ;mal 2
ASL      ;mal 2
ASL      ;mal 2
ASL      ;mal 2
```

Nun steht das Ergebnis im Akku und Sie können damit weiter operieren. Komplizierter wird das aber schon, wenn Sie nicht Potenzen von 2, sondern - sagen wir mal - von 3 oder 5 benötigen. Besser und auch schneller geht das mit Tabellen. Wir legen irgendwo eine Tabelle der Potenzen von 2 an:

```
TAB 1,2,4,8,16,32,64,128
```

Brauchen wir nun 2 hoch 5, dann schieben wir die Hochzahl in ein Indexregister und laden den Akku durch die indizierte Adressierung:

```
LDX #$05      ;Das ist die Hochzahl
LDA TAB,X     ;und schon ist 32 im Akku!
```

Es spielt nun auch keine Rolle mehr, ob wir die Potenzen der Zahl 2, 3 oder irgendeiner anderen Basiszahl benötigen: Tabelle anlegen, Hochzahl als Index wählen und den Akku indiziert laden. Braucht man für andere Zwecke aufeinanderfolgende Elemente der Tabelle, dann genügt es nun, durch INX oder DEX den Index zu variieren.

Startadresse($)	Label	Format	Inhalt
69D8	n320	MFLPT	320*65535
69DD	n200	MFLPT	200*65535
6FF9	scalel	1-Byte	LSB der Frequenzen
7005	scaleh	1-Byte	MSB der Frequenzen danach weitere Tabellen mit Werten zur Musikprogrammierung
849A	n32768	MFLPT	-32768
899C	fone	MFLPT 1	
89A2	logco3	MFLPT	Koeffizienten für LOG-Berechnung
89B6	sqr05	MFLPT	SQR(1/2)
89BB	sqr20	MFLPT	SQR(2)
89C0	neghlf	MFLPT	-0.5
89C5	log2	MFLPT	ln 2
8B2E	tenc	MFLPT	10
8E17	n0999	MFLPT	99 999 999.9
8E1C	n9999	MFLPT	999 999 999
8E21	nmil	MFLPT	1 000 000 000
8F76	fhalf	MFLPT	0.5
9005	logeb2	MFLPT	1/ln2
900B	expco7	MFLPT	Koeffizienten für EXP-Berechnung
9485	pi2	MFLPT	Pi/2
948A	twopi	MFLPT	2*Pi
948F	fr4	MFLPT	0.25
9495	sinco5	MFLPT	Koeffizienten für SIN,COS,...
94AE	sinco0	MFLPT	2*Pi
94E4	atnc11	MFLPT	Koeffizienten für ATN-Berechnung
951B	atnc00	MFLPT	1
9F29	angval	2-Byte	Sinuswerte 0 bis 90 Grad in 10 Grad-Schritten

Tabelle 4. Die wichtigsten Zahlentabellen im C 128-ROM mit ihren entsprechenden Startadressen

Startadresse($)	Inhalt
A004	CBMBASIC
A09E	Texte der Basic-Befehlsworte (im letzten Byte ist jeweils das Bit 7 gesetzt)
A19E	Texte der Basic-Fehler- und Systemmeldungen (im letzten Byte ist Bit 7 gesetzt)
A364	Weitere Systemmeldungen:OK,ERROR,... (das letzte Byte ist jeweils 0)
ACFC	Fehlermeldungen für INPUT (letztes Byte ist 0)
E460	BASIC BYTES FREE
E473	Einschaltmeldung
ECE6	LOAD <RETURN>,RUN <RETURN>
F0BD	Texte für Ein- und Ausgabe-Operationen
FD10	CBM80

Tabelle 5. Die wichtigsten Texttabellen im C 64-ROM mit ihren entsprechenden Startadressen

Startadresse($)	Label	Inhalt
41BB	sigmsg	Systemmeldung bei Kaltstart
4417	reslst	Liste der Basic-Befehlsworte (Bit 7 des letzten Byte ist jeweils gesetzt)
484B	errtab	Liste der Fehlermeldungen (Bit 7 des letzten Byte ist jeweils gesetzt)
63F5		Namen der Programmautoren
A7E8		ARE YOU SURE?
CEB2	pky1	Standardtexte der Funktionstasten
F6B0	msgtbl	Kernel-Textmeldungen
F90B		BOOTING

Tabelle 6. Die wichtigsten Texttabellen im ROM des C 128 mit ihren entsprechenden Startadressen

Komplexe Tabellen

Diese einfachste Art der Ansteuerung einer Tabelle hat natürlich gewisse Einschränkungen zur Folge: Die Elemente dürfen nicht größer als 255 (also 1 Byte) sein, es dürfen nicht mehr als 256 Elemente verwendet werden.

Hätte unsere Potenztabelle nun immer 16-Bit-Werte aufgelistet, gehörten also zu jedem Element 2 Byte, dann müßte der Index vor dem Zugriff in die Tabelle jeweils verdoppelt werden. Dazu wieder unsere Tabelle der Zweierpotenzen als Beispiel:

```
0.1, 0.2, 0.4, 0.8, 0.16, 0.32, 0.64, 0.128
```

Hier haben wir die Potenzwerte jeweils in der Reihenfolge MSB,LSB abgelegt. Suchen wir nun den Wert für 2 hoch 5, dann programmieren wir:

```
LDA #$05 ;Das ist wieder die Hochzahl
ASL      ;verdoppeln
TXA      ;und ins X-Register schieben
LDA TAB,X ;laden des MSB (10. Wert in der Tabelle)
STA ...  ;und ablegen an der Stelle, an der es
          gebraucht wird
INX      ;Index erhöhen
LDA TAB,X ;laden des LSB
```

Startadresse($)	Label	Inhalt
46FC	stmdsp	Adressentabelle der Basic-Befehle
AE63	kydmsg	verschlüsselte Mitteilung der Programmautoren
AF00	jmptbl	Sprungtabelle der Interpreter-Routinen
C6DD	funtab	ASCII-Codes der Funktionstasten
CE74	loczp	Tabelle der Default-Werte 40-Zeichen-Bildschirm
CE8E	locabs	Tabelle der Default-Werte 80-Zeichen-Bildschirm
F7F0	config	MMU-Konfigurationen für BANK 0 bis BANK 15
FF47	kspio	Kernel-Sprungtabelle
FFF8	system	Tabelle der Systemvektoren (Initialisierung, NMI, Reset und IRQ)

Tabelle 7. Weitere wichtige Tabellen im ROM des C 128

Damit hätten wir dann die 16-Bit-Zahl aus der Tabelle gelesen. Anstelle der beiden letzten Zeilen hätte auch eine einzige genügt:

```
LDA TAB+1,X ;laden des LSB
```

Adressen sind solche 16-Bit-Werte und daher findet man diese Technik der Tabellenmanipulation auch sehr häufig bei Adressentabellen. Beispielsweise haben wir im ersten Modul des Programms 30 ab Zeile 970 auf diese Weise eine Sprungadresse aus der Tabelle SPRTAB gelesen. Dazu werden wir gleich noch kommen.

Es gibt Tabellen, deren Elemente jeweils mehr als 2 Byte enthalten. In solchen Fällen genügen häufig zwei oder mehrere ASL des Index oder aber man führt jeweils eine Addition des entsprechenden Offset zum Index aus.

Lange Tabellen

Einige Tabellen, besonders Texttabellen, sind länger als 256 Byte. In dem Fall ist es nicht mehr möglich, die einzelnen Elemente (oder Teile der Elemente) mittels der bisher angewandten absolut X-indizierten (oder auch Y-indizierten) Adressierung anzusprechen, denn die Register fassen nur Zahlen von 0 bis 255. Wir greifen dann zur indirekt-indizierten Adressierung. Ein 16-Bit-Zeiger in der Zeropage wird mit der Startadresse der Tabelle geladen, das Y-Register dient als Index. Das Ansprechen der einzelnen Bytes geschieht dann beispielsweise wie folgt:

```
LDA  INDEX  ;aktuellen Index laden
ASL         ;und verdoppeln (Elemente sind 2-Byte-
             Werte)
BCC  KLEIN  ;verzweigen, wenn dabei kein Überlauf
             eintrat
INC  ZERO+1 ;bei Überlauf MSB der Tabellen-
             Startadresse erhöhen
TAY         ;Offset ins Indexregister schieben
```

Inhalt		Ziellabel	Funktion
JMP	$84B4	ayint	FAC —> Integer mit Vorzeichen
JMP	$793C	givayv	Integer in Y/A zu FLPT in FAC
JMP	$8E42	fout FAC	—> String, Adresse in A/Y
JMP	$8052	val1	String auswerten
JMP	$8815	getadr	FAC —> Integer in Y/A
JMP	$8C75	floatc	Exponent in FAC, normalisieren
JMP	$882E	fsub	FAC = FAC - (A/Y)
JMP	$8831	fsubt	Basic-Funktion Minus
JMP	$8845	fadd	FAC = FAC + (A/Y)
JMP	$8848	faddt	Basic-Funktion Plus
JMP	$8A24	fmult	FAC = FAC * (A/Y)
JMP	$8A27	fmultt	Basic-Funktion Mal
JMP	$8B49	fdiv	FAC = (A/Y) / FAC
JMP	$8B4C	fdivt	Basic-Funktion Division
JMP	$89CA	log	Basic-Funktion LOG
JMP	$8CFB	int	Basic-Funktion INT
JMP	$8FB7	sqr	Basic-Funktion SQR
JMP	$8FFA	negop	Basic-Funktion negatives Vorzeichen
JMP	$8FBE	fpwr	
JMP	$8FC1	fpwrt	Basic-Funktion Potenz
JMP	$9033	exp	Basic-Funktion EXP
JMP	$9409	cos	Basic-Funktion COS
JMP	$9410	sin	Basic-Funktion SIN
JMP	$9459	tan	Basic-Funktion TAN
JMP	$94B3	atn	Basic-Funktion ATN
JMP	$8C47	round	FAC runden
JMP	$8C84	abs	Basic-Funktion ABS
JMP	$8C57	sign	Vorzeichenflag —> Akku
JMP	$8C87	fcomp	FAC mit (A/Y) vergleichen
JMP	$8437	rnd0	Zufallszahl holen
JMP	$8AB4	conupk	(A/Y) —> FAC
JMP	$8A89	romupk	(A/Y) —> ARG
JMP	$7A85	movfrm	(A/Y) —> FAC
JMP	$8BD4	movfm	(A/Y) —> FAC
JMP	$8C00	movmf	FAC —> (X/Y)
JMP	$8C28	movfa	ARG —> FAC
JMP	$8C38	movaf	FAC —> ARG
JMP	$4828	optab	Tabelle der Prioritätsflags der mathematischen Routinen
JMP	$9B30	drawln	Strecke zeichnen
JMP	$9BFB	gplot	Punkt setzen
JMP	$6750	cirsub	Drehung ausführen
JMP	$5A9B	run	Basic-Statement RUN
JMP	$51F3	runc	Basic-Zeiger initialisieren, CLR
JMP	$51F8	clear	Basic-Statement CLR
JMP	$51D6	new	Basic-Statement NEW
JMP	$4F4F	lnlprg	berechnen der Linkadressen
JMP	$430A	crunch	Wandlung von Text in Tokens
JMP	$5064	fndlin	
JMP	$4AF6	newstt	Stoptaste abfragen, nächsten Basic-Befehl holen
JMP	$78D7	eval	Ausdruck auswerten
JMP	$77EF	frmevl	folgenden Ausdruck auswerten
JMP	$5AA6	runprg	aktives Programm starten
JMP	$5A81	setexc	Programm-Modus setzen
JMP	$50A0	linget	Zeilennummer holen
JMP	$92EA	garba2	Garbage collection ausführen
JMP	$4DCD	execln	

Tabelle 8. Die C 128-Sprungtabelle der Interpreter-Routinen

```
LDA (ZERO),Y;in ZERO und ZERO+1 liegt die
             Startadresse der Tabelle
            ;und wir haben das LSB eines Elementes
             geladen
STA ...     ;an geeigneter Stelle speichern
INY         ;Indexregister auf MSB richten
LDA (ZERO),Y;MSB laden
STA ...     ;und an geeigneter Stelle
             weiterverwenden
```

Dabei war ZERO/ZERO+1 der Vektor in der Zeropage, der auf den Tabellenstart wies und INDEX eine Speicherstelle, die den gerade aktuellen Index enthielt, beispielsweise die Hochzahl bei einer Potenztabelle. Noch mehr Möglichkeiten bieten sich, wenn man für den Index einen 16-Bit-Wert reserviert. Im folgenden Beispiel seien INDEX/INDEX+1 die dafür gedachten Speicherstellen:

```
LDA  INDEX   ;LSB des Index laden
ASL          ;und verdoppeln (Elemente sind 2-Byte-
              Werte)
TAY          ;Offset ins Indexregister schieben
LDA  INDEX+1 ;MSB des Index laden
ROL          ;Ebenfalls verdoppeln, aber mit Carry-Bit
ADC  ZERO+1  ;dazu MSB der Tabellenadresse addieren
STA  ZERO+1  ;und als neues MSB merken
LDA (ZERO),Y ;jetzt LSB des aktuellen Elementes laden
STA  ...
INY           ;Indexregister auf MSB richten
LDA (ZERO),Y  ;und MSB des Elementes laden
STA  ...
```

Auf diese oder ähnliche Weise können Sie noch so ausgedehnte Tabellen beherrschen.

Im Vergleich zu Zahlen- oder Adreßtabellen weisen Texttabellen meist die Besonderheit von Elementen variabler Bytezahl auf. Beim Lesen der einzelnen Bytes eines Elements fügt man hier immer eine Prüfung auf ein Textende-Merkmal ein. Solche Merkmale sind beispielsweise Null-Bytes. Durch ein BEQ kann dann reagiert werden und zwei Null-Byte markieren das Ende der Tabelle. Manchmal verwendet man auch etwas platzsparendere Kennzeichen wie ein gesetztes Bit 7 eines Zeichens. Dann darf allerdings die Tabelle keine Zeichen enthalten, die von sich aus schon mit gesetzten Bit 7 aufwarten. Hier wird dann durch BMI oder BIT und nachfolgendes Abfragen der entsprechenden Flaggen geprüft, ob ein Textende-Merkmal vorliegt.

Adressentabellen

Das Lesen von Adressentabellen haben wir vorhin bei den Zahlentabellen schon mitbehandelt. Sie verhalten sich wie Tabellen mit 2-Byte-Elementen. Hier soll es nun darum gehen, wie man die so gefundenen Adressen weiterverwendet, um einen Sprung an die herausgesuchte Adresse zu vollziehen. Die Technik der selbstmodifizierenden Sprunganweisung haben wir im Programm 30 in Zeile 1070 gewählt. Die Zeilen 1000 bis 1030 lesen LSB und MSB der Zieladresse aus der Sprungtabelle und tragen sie hinter die JSR-Anweisung in Zeile 1070 ein. Dorthin gelangt danach das Programm und vollzieht den Sprung.

Der Nachteil dieser Technik ist, daß sie nur in RAM-Bereichen funktioniert, weil ins Programm geschrieben werden muß. Arbeitet man mit ROMs oder EPROMs, dann bieten sich zwei andere Möglichkeiten an, von denen wir zuerst die Verwendung eines indirekten Sprungs vorstellen wollen. Dazu speichert man die gelesenen Tabellenwerte in einen Vektor aus der Zeropage (beispielsweise ZWSP/ZWSP+1) und springt dann mit

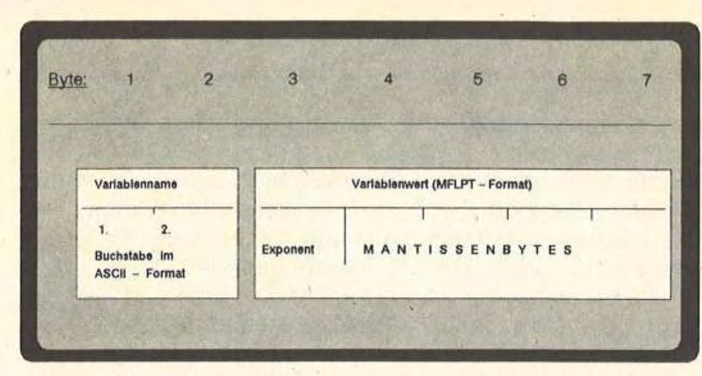


Bild 32. So wird eine Fließkommavariable in die Variablentabelle eingetragen. Byte 1 und 2 enthalten den Namen

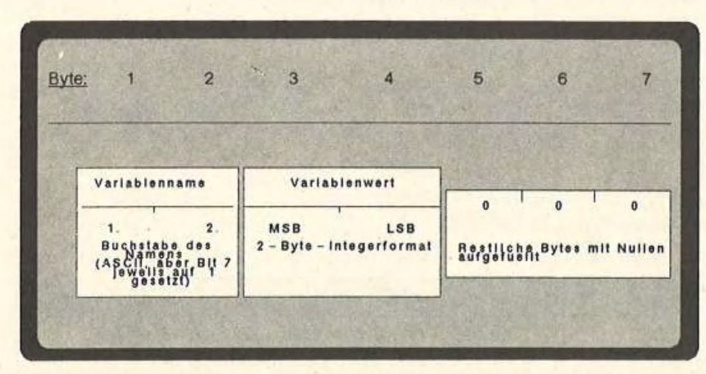


Bild 33. Format eines Integervariablen-Eintrages in die Variablentabelle. Byte 5, 6 und 7 bleiben ungenutzt.

```
JMP (ZWSP) ;das ist der selten benutzte indirekte
            Sprung
```

in die gesuchte Routine. Nebenbei bemerkt: ZWSP/ZWSP+1 muß nicht unbedingt in der Zeropage stehen: Man kann beliebige andere Speicherbereiche für diesen Vektor verwenden.

Auf den ersten Blick etwas irritierend wirkt die andere Technik, die sich des Stapels bedient. Hier ein Beispiel:

```
LDA INDEX ;aktuellen Index laden
ASL       ;und verdoppeln (Adresstabelle!)
TAX       ;ins Indexregister schieben
INX       ;Indexregister auf MSB richten
LDA TAB,X ;MSB der Zieladresse laden
PHA       ;und auf den Stapel schieben
DEX       ;Indexregister auf LSB richten
LDA TAB,X ;LSB der Zieladresse laden
PHA       ;und auf den Stapel schieben
RTS       ;!!!
```

Die Frage ist: Was macht RTS? Hier die Antwort und gleichzeitig die Lösung des Rätsels:

1) RTS holt die auf dem Stapel gespeicherte Adresse ab und schreibt sie in den Programmzähler. Damit die Reihenfolge LSB/MSB stimmt, muß als letztes das LSB im Stapel landen.

2) RTS vermindert dann den Stapelzeiger um 2. Das sei nur der Vollständigkeit halber gesagt.

3) RTS addiert zum Programmzähler eine 1 und dann läuft das Programm von dieser Adresse an weiter.

Insgesamt ergibt sich daraus dann ein Sprung zum gewünschten Programm. Wegen des dritten Punktes der RTS-Tätigkeit muß man aber darauf achten, daß in der Adressentabelle nicht ZIELADRESSE, sondern immer ZIELADRESSE-1 steht!

Mir wird bei diesem Sprung über den Stapel immer etwas mulmig zumute. Allzu unklar ist der Gebrauch des RTS. Ich bin mir nie so ganz sicher, ob ich (oder ein anderer Benut-

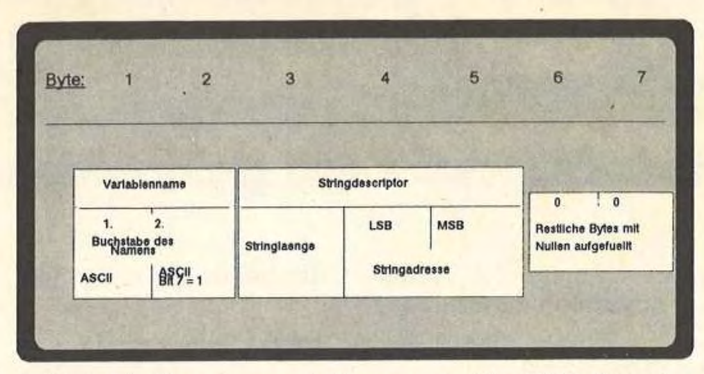


Bild 34. Ein String erzeugt diesen gegenüber Bild 32 und 33 komplexeren Eintrag in die Variablentabelle

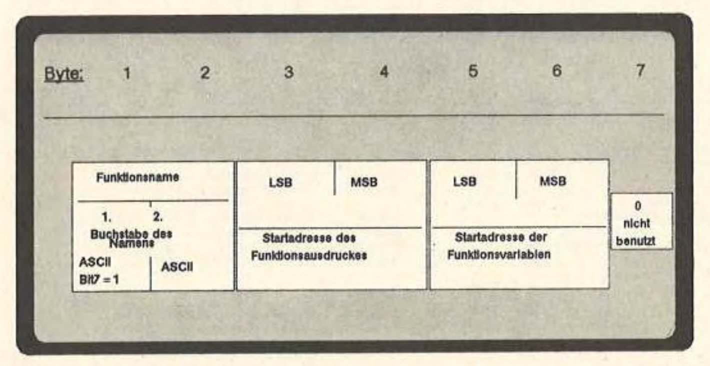


Bild 35. Auch das findet sich in der Variablentabelle: eine durch den Benutzer definierte Funktion.

zer) nach einigen Monaten ein Programm mit diesem Trick noch völlig durchschauen kann.

Nun wenden wir uns besonderen Tabellen zu, nämlich den Variablentabellen, die der Basic-Interpreter anlegt. Wozu das, werden Sie vielleicht fragen, wir programmieren doch in Assembler? Nun, es sei keinem verwehrt, sich das Leben unnötig schwerzumachen! Wer aber ökonomisch programmieren möchte, dem lege ich nicht nur die Routinen des Basic-Interpreters, sondern auch den problemlosen Umgang mit Variablen durch diesen Interpreter ans Herz.

Kooperation von Basic und Assembler

Nehmen wir einmal an, wir schreiben ein Assemblerprogramm, das alle Variablentypen und auch Arrays benötigt und diese während des Programmlaufes erst erhält (durch manuelle Eingabe, von Diskette etc.). Was hätten wir zu programmieren? Handelt es sich nicht nur um ganz wenige Werte (für die braucht man keinen großen Aufwand zu treiben), dann muß eine Routine geschrieben werden, die die Abfrage durchführt (beispielsweise mit einer Aufforderung an den Benutzer, den Wert nun einzutippen). Weiterhin muß nun der Typ erkannt werden, denn beispielsweise können Integerzahlen viel einfacher und schneller verarbeitet werden als Fließkommazahlen, und wenn man Boolesche Variable auch noch zuläßt, ist wieder eine andere Behandlung angesagt – von Strings oder Arrays der verschiedenen Typen sowie Funktionsdefinitionen ganz zu schweigen. Damit aber noch nicht genug! Die eingegebenen Werte müssen irgendwo so sinnvoll abgelegt werden, daß sie im richtigen Format jederzeit schnell wiedergefunden werden können, Fehler müssen aufgefangen und eventuelle Ausgabemöglichkeiten eingeplant werden: Eine wahre Herkulesaufgabe!

Wie leicht haben wir es da in Basic, wo all dies der Interpreter mit seinen Routinen für uns erledigt. Außer in wenigen Spezialfällen verfahre ich daher meistens so: Ein Basic-Rahmen-Programm erledigt die Annahme und Organisation (fast) aller Variablen und Arrays. Aus diesem Programm wird dann in das Assemblerprogramm geschaltet, das mit den eingegangenen Werten arbeitet. Auf diese Weise spielt sich der von der Geschwindigkeit her kritische Teil eines Programms in der schnellen Maschinensprache ab, der von daher aber unkritische Teil der Variablenorganisation (häufig dreht es sich ja um einen interaktiven Teil) im Rahmen des Basic und höchst einfach. Um so arbeiten zu können, müssen wir mehr über die Variablentabellen wissen und auch über die Routinen, die der Interpreter zum Zugriff darauf anbietet.

Variablentypen des Basic

Sehen wir uns zunächst einmal die verschiedenen Arten von Variablen des Basic an: Eine erste grobe Unterteilung liefert zwei Sorten von Variablen. Man findet nämlich sogenannte indizierte und nichtindizierte. Die indizierten sind solche, die in einem Zusammenhang mit anderen indizierten in einer bestimmten Ordnung, dem Feld oder Array stehen und die durch einen Index voneinander unterschieden werden (beispielsweise A(2), A(7) und so fort). Ihnen werden wir uns später widmen. Es bleiben also zunächst die Variablen ohne Index, von denen wir in der durch den Interpreter angelegten Variablentabelle vier Sorten finden. Jede Sorte beansprucht einen sieben Byte langen Eintrag in der Tabelle.

Am häufigsten verwendet der Basic-Programmierer (und der Assemblerspezialist wohl auch) die Fließkommavariable. Was man darunter zu verstehen hat, haben Sie ja inzwischen erfahren. Im Basic-Programmtext tauchen diese Variablen ohne weitere Kennung auf, beispielsweise als A, A1, CD und so fort. Bild 32 zeigt Ihnen den Aufbau eines solchen Fließkomma-Variablen-Eintrages in die Variablentabelle. Die beiden ersten Byte enthalten den Namen (im ASCII-Format), die restlichen fünf Byte den Wert der Variablen im MFLPT-Format.

Integervariable (also ganze Zahlen, die sich in zwei Byte ausdrücken lassen) werden im Basic-Text durch das %-Zeichen markiert. In Bild 33 sehen Sie den Unterschied zur Fließkommavariablen beim Eintrag in die Variablentabelle. Auch hier geben die beiden ersten Byte den Namen der Variablen wieder, dabei findet zwar das ASCII-Format Anwendung, aber bei beiden Byte ist als Kennung noch Bit 7 gesetzt. Die Bytes 3 und 4 enthalten den 2-Byte-Variablenwert in der Reihenfolge MSB/LSB. Die restlichen Byte sind mit Nullen gefüllt, sie bleiben unbenutzt.

Wie Sie sicher wissen, sind Stringvariablen durch das $-Zeichen markiert. Ihr Eintrag in die Variablentabelle ist etwas komplexer als die beiden bisher betrachteten Typen, siehe Bild 34. Die beiden ersten Byte enthalten wieder den Variablennamen, wobei im zweiten Byte das Bit 7 gesetzt ist (zur Kennzeichnung des Typs). In den drei folgenden Byte findet sich der sogenannte Stringdescriptor (zu deutsch »Stringbeschreiber«). Byte 3 (das erste Byte des Descriptors) enthält die Stringlänge, die Bytes 4 und 5 die Startadresse des Textes im normalen 2-Byte-Format. Die restlichen beiden Bytes sind unbenutzt und in ihnen steht der Wert 0. In diesem Variableneintrag liegt nur eine genauere Beschreibung der Variablen! Wo also ist der Text und wie sieht er aus?

Vom oberen Ende des Basic-RAM an abwärts sind die Stringtexte zu finden. Beim C 64 also ab $A000, beim C 128

Ergänzen Sie jetzt Ihre 64'er-Sammlung

Schaffen Sie sich ein interessantes Nachschlagewerk und gleichzeitig ein wertvolles Archiv!

Kennen Sie alle Ausgaben von 64'er? Suchen Sie einen ganz bestimmten Testbericht? Oder haben Sie einen Teil eines interessanten Kurses versäumt? Suchen Sie nach einer speziellen Anwendung?

Damit Sie jetzt fehlende Hefte mit »Ihrem« Artikel nachbestellen können, finden Sie auf diesen Seiten eine Zusammenstellung aller wesentlichen Artikel der Ausgaben 01 bis 12/85.

Und so kommen Sie schnell an die noch lieferbaren Ausgaben: Prüfen Sie, welche Ausgabe in Ihrer Sammlung noch fehlt, oder welches Thema Sie interessiert. Tragen Sie die Nummer dieser Ausgabe und das Erscheinungsjahr (z.B. 2/85) auf dem Bestellabschnitt der hier eingehefteten Bestell-Zahlkarte ein. Die ausgefüllte Zahlkarte einfach heraustrennen und Rechnungsbetrag beim nächsten Postamt einzahlen. Ihre Bestellung wird nach Zahlungseingang umgehend zur Auslieferung gebracht.

Aktuell

Stichwort	Titel	Seite	Ausgabe
Allgemeines	Commodore Gestern Heute Morgen	10	01/85
Computer	Amiga — Der neue Supercomputer	8	09/85
Interview	Interview mit David Crane (Game Designer)	146	06/85
Lernen	Schule braucht Computer (VAM-Computer)	9	06/85
Messen	International Chaos Communication Congress	15	03/85
	Heiße Messe in der Wüste: CES	8	03/85
	Hannover-Messe '85	8	06/85
	Hannover-Messe '85	8	07/85
	Chicago im Zeichen der CES	8	08/85
	Aktuelles von der C'85 in Köln	15	08/85
	Btx Total (Internationale Funkausstellung)	8	10/85
	PCW-Computermesse in London	8	11/85
	Neues von der Commodore-Fachausstellung 1985	8	12/85
Recht	Die neue Abmahnmasche — Vorsicht bei Programmangeboten	8	05/85
	Die Ex-Knacker — wo sind sie geblieben?	27	08/85
	Interview mit Raubkopierern (Section 8)	28	08/85
	Schützer kontra Knacki's	23	08/85
	Raub-Talkshow	12	08/85
	Das Urheberrechtsgesetz und Gedanken zu seiner Anwendung	21	08/85
	Änderung des Urheberrechtsgesetzes	162	09/85

Buchbesprechungen

Stichwort	Titel	Seite	Ausgabe
Anfänger	Goldmann Computer Compact	87	03/85
	Basic-Wegweiser für den C 64	86	05/85
	Alles über den C 64, Sachbuchreihe, Band 1	115	06/85
	Lehrspielzeug Computer: C 64/VC 20	112	11/85
	C 64 Computerhandbuch	171	11/85
	Einführungskurs: Commodore 64	144	12/85
Anwendung	Dienstprogramme VC 20, C 64 und SX	86	06/85
	Spaß an Mathe mit dem Commodore 64	88	07/85
	Mathe für die Oberstufe mit dem C 64	88	07/85
	Mathematische Routinen VC 20, Elektrotechnik/Elektronik	112	11/85
	Commodore 64-Listings, Band 2: Dateiverwaltung, Schule, Hobby	112	11/85
	Das Trainingsbuch zum Datamat	144	12/85
C 128	Bücher zum C 128	22	10/85
DFÜ	Das Mailbox-Jahrbuch: Nutz die Netze	112	11/85
Grafik	Grafik auf dem Commodore 64 (+Fehlert. 9/85)	86	05/85
	Einführung in CAD mit dem Commodore 64	128	06/85
	Grafik & Musik auf dem Commodore 64	88	07/85
	Verschiedene Grafikbücher zum C 64	115	08/85
Programmieren	Von Basic zu Assembler: Das Commodore-Buch, Band 4	115	06/85
	64 Intern	115	06/85
	Das Interface Age System-Handbuch zum C 64	115	06/85
	Das C 64 Buch, Band 5: Simons Basic Leitfaden	144	12/85
	Basicode	144	12/85
	Noch mehr Tips und Tricks zum 64er	144	12/85
Speichern	Das Kassettenbuch zum C 64 und VC 20	87	03/85
	Die Floppy 1541 (M&T)	88	07/85
Spiele	Rombachs C 64 Spielführer	87	03/85
	Commodore 64-Listings, Band 1, Spiele	112	11/85
	35 ausgesuchte Spiele für Ihren Commodore 64	171	1/85

64'er Extra

Stichwort	Titel	Seite	Ausgabe
Prozessor	Befehlssatz des 6502/6510-Prozessors	84	09/85
Grafik	Die Videochip-Register des C 64	92	10/85
Sound	Der SID-Chip, seine Register und Programmierung	92	11/85
Speicher	Die Speicherbelegung des C 64	96	12/85

Abenteuerlösungen

Stichwort	Titel	Seite	Ausgabe
Lösungen	Dallas-Quest Lösung	90	01/85
	Guncho Krill-Enchanter ist gelöst	44	03/85
	Infocom-Geheimnisse gelüftet?	49	05/85
	Des Rätsels Lösung: Amazon	145	06/85
	Activision-Adventures entschleiert (Mindshadow, Tracer Sanction)	36	12/85
	Eureka! — ich hab's!	37	12/85
	Lösungen zu Hitchhiker's Guide und Sorcerer	39	12/85

Spiele-Tests

Stichwort	Titel	Seite	Ausgabe
007	James Bond — A View to a Kill	156	09/85
Abenteuer	Abenteuerpaket 1	48	08/85
	Shadowfire	146	09/85
	The Quest — mit C 64 auf Suche nach Drachen	47	01/85
Action	Hexenküche	50	07/85
	Master of the Lamps	48	07/85
	Rescue on Fractalus	158	10/85
	Stellar 7	49	08/85
Construction Set	Mail Order Monsters	49	08/85
	Racing Destruction Set	50	08/85
Geschicklichkeit	Australopedicus Robustus	50	08/85
	Boulder Dash II	159	10/85
	Crystal Castles	50	07/85
	Gribbly's Day out	148	09/85
	Rock'n Bolt	48	08/85
	Thing on a Spring	159	10/85
	Tom + Zaga	48	01/85
Pseudo-Adventures	Roland's Rat Race	49	08/85
	Fourth Protocol und Frankie g.t.H.	162	11/85
Renner	Die Renner 1985: Meistverkaufte Spiele	34	12/85
Schach	Viermal Schachmatt: Verschiedene Schachprogramme	32	12/85
Simulation	Elite	148	09/85
	Jump Jet	148	09/85
	Super Huey Hubschraubersimulator	49	07/85
Sport	Boxspiele: Frank Bruno's B. + Barry McGuigan Champions. B.	49	12/85
	Handkantenschlag per Joystick: Karateka + Exploding Fist	165	11/85
	Nick Faldo Plays the Open (Golf)	159	10/85
	Rallye Speedway	49	07/85
	Slapshot (Eishockey)	50	07/85
	Summer Games II	146	09/85
	World Series Baseball	49	07/85
Diverses	New York City und Air Support	145	06/85

Hardware-Tips und Bauanleitungen

Stichwort	Titel	Seite	Ausgabe
Audio/Video	Mit 5 Mark zu neuen Dimensionen (Stereoanlage am C 64)	34	05/85
	Ein Monitor ist genug (RGB+Composite an C 128)	16	10/85
C 16	Alte Datasette am C 16	31	04/85
	Alter Joystick am C 16	35	05/85
Eingabegeräte	Der Hexer — Zusatztastatur für den MSE	48	10/85
EPROM	EPROMs im Expansion-Port	46	10/85
	EPROM-Trans — Die Super-Erweiterung	42	10/85
	Das 64'er EPROM-Programmiergerät, Teil 1	44	12/85
Floppy/Datasette	Diskettenlaufwerk 1541 selbst justiert	32	10/85
	Die Datasette streikt nie wieder (Anpassung des Tonkopfs)	34	10/85
IEC-Bus	Auf zu neuen Welten: IEC-Bus im Selbstbau (+Fehlerteufel 10/85)	44	07/85
Joystick	Joystick im Selbstbau	33	03/85
	Dauerfeuer-Adapter	46	08/85
RS232/V.24	Das 30-Mark-Interface (Selbstbau RS232)	29	03/85
	Genau betrachtet: Die RS232/V.24-Schnittstelle	80	05/85
Diverses	Userport-Display	36	05/85
	Reset-Taster für alle Fälle (+Fehlert. 9/85)	130	06/85
	Aus eins mach vier (absturzfreie Betriebssystemumschaltung)	41	07/85

Hardware-Grundlagen

Stichwort	Titel	Seite	Ausgabe
Computer	Was bringt der C 128?	28	11/85
Drucker	Welcher Drucker ist der Richtige? (Grundlagen)	15	05/85
	Hammerwerke — wie funktionieren Typenraddrucker	32	06/85
	Die Alternativen: Thermo-, Tintenstrahldrucker + Plotter	24	07/85
Eingabegeräte	Versteht Sie Ihr Computer? (Wie funktionieren Eingabegeräte)	44	09/85
Floppy	Floppy oder Datasette?	129	06/85
Monitore	Wie funktionieren sie, was ist beim Kauf zu beachten?	16	12/85
	Das Kabel zum Monitor: Welche Normen gibt es?	28	12/85
Peripherie	Grafikeingabegerät: Wie funktionieren sie?	30	08/85

Hardware-Tests

Stichwort	Titel	Seite	Ausgabe
Computer	Generationswechsel: Test C 16	16	01/85
	Erster ausführlicher Test C 128 PC (Teil 1)	16	06/85
	Erster ausführlicher Test C 128, PC (Teil 2)	17	07/85
DFÜ	Marktübersicht Modems & Akustikkoppler	32	07/85
Drucker	Vergleich: Drucker unter 700 Mark (Tests und Marktübersicht)	18	05/85
	Tests und Marktübersicht Typenraddrucker	35	06/85
	Test: Brother EP 44	27	07/85
	Brother TC-600	118	08/85
	Riteman C+	133	09/85
	Panasonic KX-P1091	134	09/85
	Star SG 10C	132	09/85
	Melchers CP-80X — wie hätten Sie's denn gern?	25	10/85
	Geheimtip: Der RFI DP 165	24	10/85
	Epson GX 80 — einer für alle	26	10/85
	MPS 803 — ein Drucker für alle Gelegenheiten?	40	1/85
	Epson JX-80 das vielfarbige Druck-Genie	38	11/85
	Epson FX-85 neue Referenz	42	11/85
	SP 1000 VC — Superstar mit Haken	41	11/85
	Der NEC-P2 — das fernöstliche Wunder	159	12/85
	DMPG9 — eine solide Sache	162	12/85
	Das Doppelleben des Joystick-Ports: 10er-Tastaturen	50	09/85
	Joysticks: Test und Marktübersicht (+Fehlerteufel 12/85)	19	11/85
	Es geht auch anders: Lightpens und Trackballs	22	11/85
EPROMer	Frisch gebrannt ist halb gespeichert (EPROM-Programmiergeräte im Test)	39	07/85
	QuickByte II — das Kraftpaket	14	10/85
Floppy/Datasette	Turbo-Floppies, zweite Generation: Speeddos plus + Prologic DOS	28	10/85
	Das große Rennen: Schnelle Bandlaufwerke	37	10/85
	Professionelle Floppylaufwerke für den C 64 (IEC-Floppies)	30	10/85
	Gut gekauft ist halb gespeichert (Marktübersicht Disketten)	38	10/85
Grafik	Die Videowerkstatt (Digitizer-Test)	32	05/85
	Digitalbilder m.d. C 64: PrintTechnik Digitizer	24	01/85
Interface	Hardware-Interface ganz weich: Test EC 64	23	01/85
	Gute Connections — Übersicht Schnittstellen	21	03/85
	Card/Print +6 — Das Allround-Interface	20	03/85
	Das Wiesemann-Centronics-Interface	18	03/85
	Erst ein IEC-Bus öffnet Tür und Tor (+Fehlert.4/6-85)	24	03/85
Monitore	Marktübersicht: Monochrome Monitore	30	12/85
Musik	Trommelwirbel: Test Digital Drums	45	08/85
	Die Musikhardware zum C 64	17	09/85
Roboter	Roboter selbst gebaut (Fischertechnik)	167	10/85
Scanner	So lernt Ihr Drucker lesen	30	06/85
Speicher	Speichertuning VC 20: Test 64 KByte Karte	26	01/85
Steuern	Flottes Türmchen: MEA-Interface	116	08/85

Kurse

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler ist keine Alchimie, Teil 5	142	01/85
	Assembler ist keine Alchimie, Teil 7	124	03/85
	Assembler ist keine Alchimie, Teil 9	138	05/85
	Assembler ist keine Alchimie, Teil 10	127	07/85
	Assembler ist keine Alchimie, Teil 11	126	08/85
	Assembler ist keine Alchimie, Teil 12	109	09/85
	Assembler ist keine Alchimie, Teil 13 (Schluß)	143	10/85
C 128	Entdeckungsreise duch den C 128	42	12/85
Effektives Programieren	Müllabfuhr im Computer: Garbage Collection, Teil 1	122	01/85
	Finden mit System, eine neuartige Suchmethode, Teil 3	148	03/85
	Sortieren mit dem Computer, Teil 2	159	05/85
	Sortieren mit dem Computer, Teil 3	124	06/85
	Sortieren mit dem Computer, Teil 4	138	08/85
	Sortieren mit dem Computer, Teil 5	124	09/85
	Sortieren mit dem Computer, Teil 6 (Schluß)	150	12/85
Extern	C 64 extern — Der Weg nach draußen, Teil 1	144	08/85
	C 64 extern — Der Weg nach draußen, Teil 2	122	09/85
	C 64 extern — Der Weg nach draußen, Teil 3 (Schluß)	129	10/85
Floppy	In die Geheimnisse der Floppy eingetaucht, Teil 4	148	01/85
	In die Geheimnise der Floppy eingetaucht, Teil 5	130	03/85
	In die Geheimnise der Floppy eingetaucht, Teil 6	145	05/85
	In die Geheimnisse der Floppy eingetaucht, Teil 7 (Schluß)	116	06/85
	Directory-Manipulationen I	140	06/85
Floppy	Directory-Manipulationen II	163	10/85
Grafik	Hires 3 — 15 neue Basic-Befehle, Teil 2	136	03/85
	Hires 3 — Grafikkurs-Anwendung, Teil 3 (Schluß)	152	08/85
	Sprites ohne Geheimnisse	40	08/85
	Streifzüge durch die Grafikwelt, Teil 1	106	09/85
	Streifzüge durch die Grafikwelt, Teil 2	149	11/85
Logeleien	Logeleien, Teil 1	143	07/85
	Logeleien, Teil 2	136	08/85
	Logeleien, Teil 3 (Schluß)	115	09/85
Musik	Dem Klang auf der Spur, Teil 2	136	01/85
	Dem Klang auf der Spur, Teil 4	131	04/85
	Dem Klang auf der Spur, Teil 5	152	05/85
	Dem Klang auf der Spur, Teil 7	132	07/85
	Dem Klang auf der Spur, Teil 8	133	08/85
	Dem Klang auf der Spur, Teil 9	126	10/85
	Dem Klang auf der Spur, Teil 10 (Schluß)	157	11/85
Speicher	Memory Map mit Wandervorschlägen, Teil 3	126	01/85
	Memory Map mit Wandervorschlägen, Teil 5	144	03/85
	Memory Map mit Wandervorschlägen, Teil 7	120	06/85
	Memory Map mit Wandervorschlägen, Teil 8	140	07/85
	Memory Map mit Wandervorschlägen, Teil 9	129	08/85
	Memory Map mit Wandervorschlägen, Teil 10	112	09/85
	Memory Map mit Wandervorschlägen, Teil 11	133	10/85
	Memory Map mit Wandervorschlägen, Teil 12	145	11/85
	Memory Map mit Wandervorschlägen, Teil 13	146	12/85
Sprachen	Basic ist out — es lebe Forth	43	01/85
VC 20	Der gläserne VC 20, Teil 4	130	01/85
	Der gläserne VC 20, Teil 6 (Schluß)	155	03/85

Software-Tips

Stichwort	Titel	Seite	Ausgabe
C 128	Erste Fragen und Antworten zum C 128	14	09/85
	Fragen und Antworten zum 128er	20	10/85
	Fragen und Antworten zum 128er	40	12/85
Drucker	Der MPS 802 lernt Deutsch	30	05/85
	Centronics-Interface für jeden Bedarf	78	07/85
Textverarbeitung	Software Corner — professionelle Programme richtig eingesetzt (Vizawrite-Tips)	174	12/85
Tips & Tricks	Autoboot beim C 64	86	03/85
	Verbindungsfreundlich (Parallelschnittstelle des VC 20)	91	03/85
	Undefinierte Opcodes des 6502	84	03/85
	Durch POKEs zum Erfolg (Spiele-POKEs)	83	03/85
	Tips und Erweiterungen zu Hi-Eddi und Simons Basic	88	03/85
	Basic-Befehle im Griff	79	05/85
	Durch POKEs zum Erfolg: Spiele-POKEs	78	06/85
	Formatierte Eingabe	148	06/85
	Hi-Text (Text in Hires)	70	08/85
	Verbotene Variablen	66	09/85
	Verschiedene Routinen für Anfänger und Profis (+Fehlerteufel 12/85)	88	11/85
	Der Trick mit dem Joystick (Joystickabfrage)	24	11/85
	Verschiedene Tips für Anfänger und Fortgeschrittene	106	12/85

Software-Grundlagen

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler? Assembler! (Einführung)	32	01/85
	Assembler-Bedienung leicht gemacht, Teil 1	169	12/85
DFÜ	Der erste Kontakt mit DFÜ	40	06/85
	Die Netze der Post: Btx, Datex-P, Telebox	46	06/85
	DFÜ — Was ist das?	44	06/85
	Mailbox für Anfänger	30	07/85

Stichwort	Titel	Seite	Ausgabe
Datei	Die wichtigsten Begriffe der Dateiverwaltung	42	05/85
	Dateiverwaltung ist nicht gleich Datenbank	44	05/85
	Dateiverwaltung: Was Sie beim Kauf beachten sollten	40	05/85
Drucker	Hardcopy leicht gemacht (wie programmiert man Hardcopies)	34	09/85
EPROM	Wie sage ich es meinem EPROM? (EPROM-Grundlagen)	35	07/85
Funktionen	Funktionen für Anfänger	164	05/85
Lernen	Besser lernen mit dem Computer	166	10/85
Musik	Klangprogrammierung ohne Ballast	19	09/85
Spiele	Taktik- und Strategiespiele	46	03/85
	Play by Mail und Play by Modem	153	09/85
Sprachen	Sprachen für Computer, Teil 2	46	05/85
Textverarbeitung	Von der Schreibmaschine zum Textsystem	34	03/85

Listings zum Abtippen

Stichwort	Titel	Seite	Ausgabe
Anwendung	Der C 64 als Handballtrainer (AdM)	52	01/85
	Ligatab — ohne Organisation kein Tor (LdM)	50	03/85
	Gut Ziel mit dem C 64 — Schützenvereinsergebnisse (AdM)	52	03/85
	Weißt du, wieviel Sternlein stehen (Sternkarte) (AdM) (+Fehlert. 6/85)	52	05/85
	Haushaltsbuchführung (AdM)	52	07/85
	Netzwerkanalyse: Ein Programm für Hobbyelektroniker (AdM)	52	08/85
	Prüfungsfragen (AdM)	52	09/85
	Fit in Latein mit dem C 64 (AdM)	52	10/85
	Lyrik-Maschine (AdM)	52	11/85
	Hypra-Platos (LdM)	50	11/85
	Der Chemie-Assistent (AdM)	52	12/85
	SMON Teil 3: Ohne gutes Werkz. geht es nicht	69	01/85
	Hypra-Ass (LdM)	51	07/85
	Neues vom SMON (+Fehlerteufel 11/85)	87	10/85
	Reassembler zu Hypra-Ass (+Fehlerteufel 12/85)	97	11/85
	Ergänzungen zu Hypra-Ass (bedingte Verzweigungen)	96	11/85
	Tips & Tricks zum SMON (inklusive Diskmonitor)	100	12/85
Bildschirmseite	Auflösung Wettbewerb Bildschirmseite: Drei Top-Programme	158	09/85
DFÜ	Terminalprogramm der Spitzenklasse (+Fehlerteufel 10/85)	149	07/85
Datei	SMU — Der Maskengenerator (LdM)	50	12/85
Drucker	Hi-Eddi-Druckerroutinen	69	06/85
	C 64 Schreiberling — Drucken wie gemalt	54	10/85
	Koalabilder Farbhardcopy auf Epson JX-80	39	11/85
Einzeiler	Die nächsten 14 aus d. Einzeilerwettbewerb	157	01/85
Floppy	Hypra-Load mal 4 (+Fehlerteufel 3/85)	82	01/85
	Diskettenmonitor	83	08/85
	Disk-Designer	70	09/85
	Herzoperation (Hypra-Load + Hypra-Ass + DOS5.1 + Centronics)	104	11/85
Grafik	Vier Pseudo-VICs mit 32 Sprites	76	01/85
	Hi-Eddi: Zeichen- und Malproggramm (LdM)	50	01/85
	Elektrotechnisches Zeichnen mit dem VC 20	71	03/85
	Mini-Grafik VC 20, Grafikhilfe	69	05/85
	Trickfilm mit dem C 64: Bewegte 3D-Grafik (LdM) (+Fehlerteufel 6/85)	51	05/85
	Kurvenplotten mit Hardcopy auf dem C 16	68	06/85
	Doppelte Grafikauflösung für C 128	33	11/85
	Bilder aus einer anderen Dimension (Apfelmännchen)	80	11/85
Intelligenz	VIC — das intelligente Programm (Wettbewerbssieger)	173	05/85
Musik	Sound Machine (+Fehlerteufel 10/85)	23	09/85
	Sound Master (Basic-Erweiterung)	31	09/85
Spiele	6510 — Die Suche nach der Prozessor	70	05/85
	Samurai (Strategiespiel)	72	06/85
	Schach dem C 64: Schachprogramm zum Abtippen	72	08/85
	Spielen auf zwei Bildschirmen: Zeichensatzscrolling (LdM)	51	09/85
	Pac-Man unter der Lupe	76	10/85
	Block Out	84	11/85
	Seekrieg per Telefon (Schiffe versenken per Modem)	82	12/85
Spielehilfe	Die Scroll-Maschine — D. Fenster zur Spielewelt (LdM) (+Fehlert. 11/85)	52	06/85
Sprachen	Tiny Forth Compiler (LdM) (+Fehlert. 9/85)	51	08/85
Textverarbeitung	Hypra-Text (LdM) (+Fehlerteufel 11/85)	50	10/85
	Drucksache — Hypra-Text, Teil 2	71	11/85
Tips & Tricks	Große Buchstaben	89	01/85
	Restore für Unterprogramme	90	01/85
Tips & Tricks	Parameterübergabe an Maschinenspracheprogramme	88	01/85
	Cursorsteuerung leicht gemacht	86	02/85
	22 Read Error — Theorie und Praxis	41	03/85
	Floppy-Lister (+Fehlerteufel 4/85)	82	03/85
	Longscreen beim VC 20	83	05/85
	C 16: Help und Trace verbessert	84	05/85
	Ordnung ist das halbe Leben (Directory-Sorter)	77	05/85
	Dokumentationshilfe, Cross-Referenz-Liste C 64 (Wettbewerb)	155	06/85
	Prost mit dem C 64: Gerätesteuerung über Userport (+Fehlerteufel 9/85)	76	06/85
	Fenster-Befehle für den C 16	84	07/85
	Elektronische Merkzettel	83	07/85
	File-Compactor	82	07/85
	REM-Killer (+Fehlerteufel 9/85)	75	07/85
	Basic-Start-Generator	74	07/85
	Komfortable Ein-/Ausgaberoutine	77	07/85
	Bildschirmmasken leicht erstellt	86	08/85
	Der Bitmap-Compander (HiRes-Bilder komprimieren)	81	08/85
	Hypra-Save	79	08/85
	'Procedure' — oder der C 64 kann lernen	78	08/85
	Aufgewickelt — Listingscrolling für VC 20	63	09/85
	Programmgenerator für den C 64	86	10/85
	Cross-Ref optimiert	83	10/85
	Spieletrainer: Spritekill	86	11/85
	Tipp-Utility	99	12/85
	Der EPROM-Automat (wie man Module macht)	90	12/85
	80-Zeichen-Grafik für den C 128	78	12/85
	Hyper Screen (Sprites auf dem Bildschirmrand)	76	12/85
Transfer	Der C 64 als PET: PET-Simulator	87	01/85
Unterprogramme	Formatierte Eingabe	156	01/85

Software-Tests

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler im Test Teil 1	34	01/85
Basic-Erweiterung	GBasic — Alles drin	28	01/85
	Macro-Basic: Die Unterprogramm-Bibliothek	137	06/85
	Darf es etwas mehr sein? — Test Business-Basic	120	08/85
	Das Intellectool	138	09/85
	Formel 64: Das Multitalent	158	12/85
DFÜ	Terminalprogramme: Übersicht	42	06/85
Datei	Vergleichstest — 7 Dateiverwaltungen auf einen Blick	118	07/85
	Aufgeräumt mit Mainfile II	157	10/85
Grafik	Malen auf dem Bildschirm (Malprogramme)	34	08/85
	Grafikprogramme auf einen Blick: Marktübersicht	38	08/85
	Vergleichstest: Grafik-Erweiterungen	37	09/85
Lernen	Softlearning — die weiche Welle des Lernens	40	01/85
	Vokabeltraining mit dem Computer	39	03/85
	Marktübersicht: Lernsoftware	168	10/85
Musik	Musik für den C 64: Übersicht Musiksoftware	26	09/85
	The Music System — Zwei auf einen Schlag	164	12/85
Sprachen	Logo — die Sprache für Einsteiger	135	05/85
	Der Ada Trainingskurs auf dem C 64	139	05/85
	Promal — die neue Sprache für Profis?	124	07/85
	Forth-wärts mit M&T-Forth 64	126	07/85
	Was leistet Pilot?	121	08/85
	Pascal für Profis (Profi-Pascal)	122	08/85
	Super-Forth 64	144	09/85
	C — die professionelle Programmiersprache für den C 64	140	09/85
	Basic 7.0 — Das Superbasic des C 128	18	10/85
	Comal 80 — die universelle Programmiersprache	151	10/85
	Turbo-Pascal auf dem C 128	30	11/85

Stichwort	Titel	Seite	Ausgabe
Textverarbeitung	Homeword - Textverarbeitung zu Hause	36	03/85
	Totl-Text — Flexibilität ist Trumpf	38	03/85
	Protext — Textprofi mit 80 Zeichen	133	05/85
	Textomat Plus kontra Vizawrite	132	06/85
	Der Preishammer (Test: StarTexter)	135	09/85
	Paperclip — ausdrücklich gut	44	11/85

So machen's andere

Stichwort	Titel	Seite	Ausgabe
Semmeln	Semmelservice mit dem C 64	147	06/85
Sport	Commodore Sportservice: Heimcomputer zur Turnierauswertung	157	07/85
Hilfe	Computer für Behinderte	182	12/85

in Bank 1 von $FF00 an. Wir sehen uns diese Aufteilungen gleich noch detaillierter an. Der Zeiger im Stringdescriptor weist genau auf das erste ASCII-Zeichen des hier gespeicherten Textes. Für den C 64 ist damit schon alles geklärt. Der C 128 aber birgt noch eine kleine Besonderheit, die die sogenannte »Garbage Collection« beschleunigt (darunter versteht man das Aufräumen von nicht mehr gebrauchten Stringtexten): Nach dem eigentlichen Text findet sich hier noch ein 2-Byte-Zeiger, der auf den Stringdescriptor weist (manchmal wird er »Codedescriptor« genannt, vermutlich deshalb, weil er sich an den Text im ASCII-Code anschließt).

Ein Außenseiter macht sich in der Variablentabelle als vierter »Variablentyp« breit: Die benutzerdefinierte Funktion. Bild 35 zeigt Ihnen solch einen Eintrag. Außer den beiden Namen-Byte zu Beginn (im ersten davon ist das Bit 7 gesetzt) finden wir hier zwei Zeiger. Der erste davon (Byte 3 und 4) weist auf die Funktionsvorschrift im Basic-Text. Der zweite Vektor enthält die Startadresse der Funktionsvariablen in der Variablentabelle (beispielsweise X aus der Funktionsdefinition DEF FN AB(X) = ...). Das letzte Byte ist unbenutzt.

Wir haben nun noch zwei Fragen zu klären: Wo befindet sich diese Tabelle und wie kann man sie benutzen? Den Ort der Variablentabelle in absoluten Adressen anzugeben ist unnötig (und beim C 64 auch nicht so ohne weiteres möglich). Dort nämlich schließt sie sich nahtlos an den Basic-Programmtext an. Beim C 128 verhält sich das einfacher: Da liegt sie in der Bank 1 gleich oberhalb der Common-Area, also ab $400.

Den genauen Anfang und die ganze Organisation all dieser Tabellen (außer mit der Variablentabelle haben wir es noch mit den Arrays und den Stringtexten zu tun) erfährt man am besten aus einer Reihe von Vektoren, die sich in der Zeropage befinden. Die Bilder 36 und 37 illustrieren die Zusammenhänge für den C 64 und für den C 128.

VARTAB heißt der erste dieser Zeiger (C 64: 45/46, C 128: 47/48). Er weist auf den Anfang der Variablentabelle und somit im allgemeinen beim C 64 auch auf das Programmtextende, beim C 128 dagegen auf $400 in Bank 1. Das Ende der Tabelle mit den einfachen Variablen erfährt man durch den nächsten Zeiger, der ARYTAB genannt wird (C64: 47/48, C 128: 49/50) und gleichzeitig auch den Anfang der Array-Tabelle verrät. STREND ist der Name des Zeigers auf das Ende der Tabelle der indizierten Variablen (C 64: 49/50, C 128: 51/52). Genaugenommen ist die in STREND gespeicherte Adresse ein Byte höher als dieses Ende.

Beide bisher genannten Tabellen wachsen zu immer höheren Adressen beim Hinzufügen weiterer Variablen oder Arrays. Anders herum – wie schon oben erwähnt – verhält sich das mit der Tabelle der Stringtexte. Diese fangen beim C 64 normalerweise direkt unterhalb des Basic-Interpreters an, also an der Adresse $9FFF, und dorthin weist der Zeiger MEMSIZ (55/56). Der C 128 stellt uns wesentlich mehr Platz zur Verfügung. Hier beginnen die Stringtexte direkt unterhalb der MMU-Register (genaugenommen unterhalb des CR = Konfigurationsregister) bei $FEFF in Bank 1. Auch hier hilft uns wieder ein Zeiger, der MAX-MEM-1 heißt und in 57/58 zu finden ist. Die aktuelle Textfront schiebt sich von String zu String immer weiter abwärts. Ihre Adresse kann man aus dem Vektor FRETOP (C 64: 51/52, C 128: 53/54) erfahren.

Beide Fronten schieben sich so im Verlauf eines Programms (wenn ständig neue Texte hinzukommen) aufeinander zu, bis sie sich irgendwann einmal berühren. Genaugenommen wird vor jedem Speichern eines neuen Textes geprüft, ob er noch in den verbleibenden Speicherabstand zwischen FRETOP und STREND paßt.

Ist das einmal nicht mehr der Fall, dann findet die Beseitigung von Stringtextmüll – die vorhin schon erwähnte Garbage Collection – statt, bei der Texte ohne Descriptor entfernt und die gültigen Texte säuberlich nach oben gestapelt werden. Reicht auch diese Maßnahme nicht mehr aus, dann meldet sich der Interpreter mit einer Fehlerbotschaft: Out of Memory Error.

Wir können nun auf die Variablentabelle zugreifen wie auf jede andere Tabelle, die Elemente zu je sieben Byte enthält. Das beigefügte Programm DUMP (Listing 34) zeigt Ihnen das für den C 64, indem es eine Liste aller definierten Variablen eines Basic-Programms und ihrer Inhalte ausgibt. Durch SYS 49152 wird diese Ausgabe gestartet. In Listing 35 finden Sie den ausführlich dokumentierten Quelltext, so daß Sie auch schnell erkennen, daß das Programm noch verbessert werden kann. Für den C 128 ist es nicht so ohne weiteres umzuschreiben, denn hier treten wieder allerlei Bank-Probleme auf.

Drei Interpreter-Routinen wurden im Programm verwendet, die mit der Ausgabe der Variableninhalte auf den Bildschirm zu tun haben: NUMDON druckt den FAC-Inhalt (hier also eine Fließkommazahl) auf den Bildschirm. Diese Routine wird durch JSR $AABC (dezimal 43708) angesteuert. OUTSTR dient zur Stringausgabe auf den Bildschirm. Dazu muß der Textstart im Vektor INDEX (das ist $22/23 oder dezimal 34/35) enthalten sein und die Länge des Stringtextes im X-Register. Sind diese Bedingungen erfüllt, dann beginnt die Ausgabe durch JSR $AB25 (dezimal 43813). LINPRT zeigt eine 2-Byte-Integerzahl auf dem Bildschirm an. Dazu muß das LSB dieser Zahl im X-Register, das MSB im Akku enthalten sein. JSR $BDCD (dezimal 48589) führt dann den Ausdruck durch.

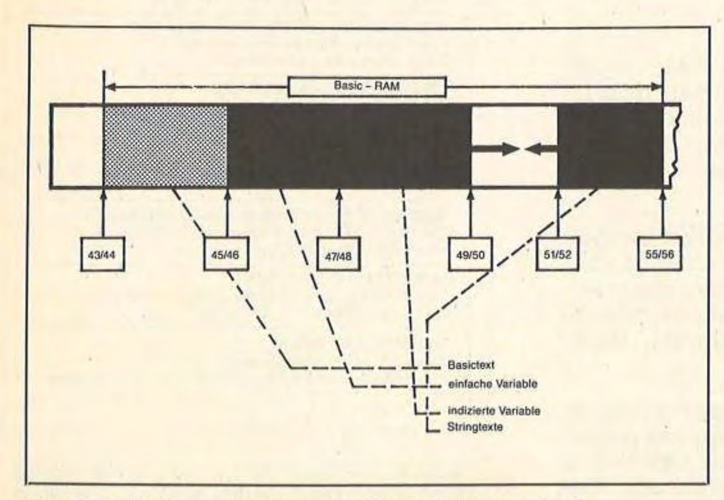


Bild 36. Die Organisation des Basic-RAM im C 64

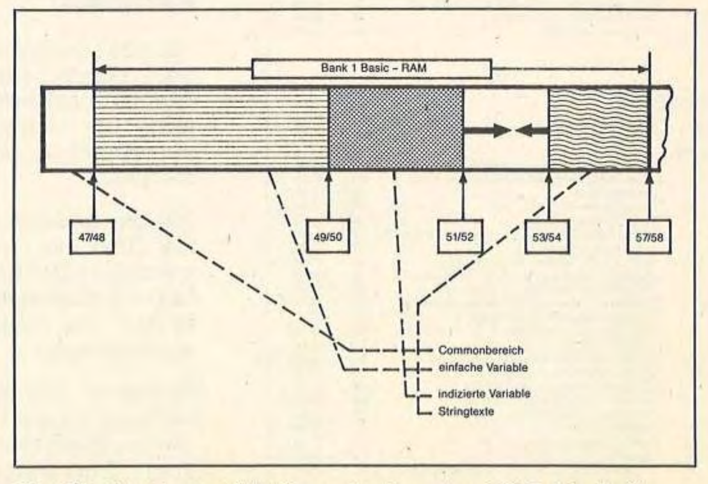


Bild 37. Die Basic-RAM-Organisation des C128 (Bank 1)

Häufiger noch ist die Aufgabenstellung, eine bestimmte Variable zu suchen. Haben wir also im Aufrufprogramm eine Variable A1 definiert, dann sollte es möglich sein, im Assemblerprogramm unter Angabe dieses Namens einen Zeiger auf den Variablenwert zu erhalten. Genau das leistet die Routine ORDVAR (C 64: $B0E7 = dezimal 45287, C128: $7B0B = dezimal 31499). Dazu trägt man den Variablennamen in die Speicherstellen VARNAM und VARNAM+1 ein (C 64: $45/6 = dezimal 69/70, C 128: $47/8 = dezimal 71/2), springt dann die Routine ORDVAR an und erhält einen Zeiger auf den Variablenwert in VARPNT (C 64: $47/8 = dezimal 71/2, C 128: $49/A = dezimal 73/4). ORDVAR ist so entgegenkommend, daß eine neue Variable automatisch eingerichtet wird, wenn die benannte noch nicht existiert.

Im Basic 7.0 des C 128 gibt es die POINTER-Funktion, mit deren Hilfe die Adresse einer beliebigen Variablen erfahren werden kann. Mittels ORDVAR läßt sich diese Funktion auch relativ einfach für den C 64 entwickeln. Ihre Kenntnisse reichen dazu allemal aus: Probieren Sie doch, diese Aufgabe zu lösen.

Eine mögliche Vorgehensweise wäre es, einen Aufruf der eigenen Pointer-Routine in der Form SYS Adresse,Variablenname+Kennung durchzuführen. Hierzu müßte dann der Text Variablenname+Kennung gelesen und in VARNAM gespeichert werden, wobei die entsprechenden Bit 7 zu setzen wären. Ein Aufruf von ORDVAR liefert dann den Zeiger auf den Variablenwert in VARPNT, der beispielsweise durch LINPRT ausgegeben werden könnte.

Es bleibt nun noch die Aufgabe, uns die andere durch den Basic-Interpreter eingerichtete Tabelle – nämlich die der Arrays – genauer anzusehen. Mit diesem Thema werden wir auch unseren Kurs beschließen.

Eine Tabelle von Tabellen, das ist der Bereich des Basic-Speichers, der den Arrays vorbehalten ist. Dem Basic-Programmierer wohlvertraut, können diese indizierten

```
Name : dump c 64              c000 c0c9
---------------------------------------
c000 : a5 2d 85 fb a5 2e 85 fc  f8
c008 : a9 0d 20 d2 ff a0 00 b1  03
c010 : fb 08 29 7f 20 d2 ff c8  74
c018 : b1 fb 08 29 7f d0 02 a9  c8
c020 : 20 20 d2 ff 28 30 0e 28  91
c028 : 30 03 4c 83 c0 a9 21 20  7b
c030 : d2 ff 4c a9 c0 28 30 28  a8
c038 : a9 24 20 d2 ff a9 20 20  64
c040 : d2 ff a9 3d 20 d2 ff a9  10
c048 : 20 20 d2 ff c8 b1 fb aa  8c
c050 : c8 b1 fb 85 22 c8 b1 fb  c7
c058 : 85 23 20 25 ab 4c a9 c0  61
c060 : a9 25 20 d2 ff a9 20 20  0c
c068 : d2 ff a9 3d 20 d2 ff a9  38
c070 : 20 20 d2 ff c8 b1 fb 48  ef
c078 : c8 b1 fb aa 68 20 cd bd  a7
c080 : 4c a9 c0 a9 20 20 d2 ff  54
c088 : a9 20 20 d2 ff a9 3d 20  26
c090 : d2 ff a9 20 20 d2 ff 18  99
c098 : c8 98 65 fb 48 a5 fc 69  fe
c0a0 : 00 a8 68 20 a2 bb 20 bc  14
c0a8 : aa a9 0d 20 d2 ff 18 a5  47
c0b0 : fb 69 07 85 fb a5 fc 69  86
c0b8 : 00 85 fc a5 fb c5 2f a5  64
c0c0 : fc e5 30 b0 03 4c 0d c0  19
c0c8 : 60 70 70 70 70 70 70 70  b8
```

Listing 34. DUMP C 64 erzeugt einen Bildschirmausdruck aller einfachen Variablen und ihrer aktuellen Werte. Bitte mit dem MSE eingeben.

```
10    -;********************************
20    -;*                              *
30    -;*           d u m p            *
40    -;*                              *
50    -;*  programm zum listen aller   *
60    -;*  variablen und ihrer werte   *
70    -;*                              *
80    -;* heimo ponnath  hamburg 1986  *
90    -;*                              *
100   -;********************************
110   -            .ba $c000
120   -;
130   -;--- verwendete labels ---------
140   -;
150   -            .eq index=$22  ;zeiger fuer outstring
160   -            .eq vartab=$2d ;start der variablentabelle
170   -            .eq arytab=$2f ;ende der variablentabelle
180   -            .eq help=$fb   ;hilfszeiger
190   -;
200   -            .eq numdon=$aabc;fac ausgeben
210   -            .eq outstr=$ab25;string ausgeben
220   -            .eq movfm=$bba2;laedt fac aus speicher
230   -            .eq linprt=$bdcd;integer ausgeben
240   -            .eq chrout=$ffd2;akkuinhalt ausgeben
250   -;
260   -;--- das programm ---------------
270   -;
280   -init       lda vartab      ;variablentabelle start
290   -           sta help        ;uebertragen
300   -           lda vartab+1
310   -           sta help+1
320   -           lda #$0d        ;carriage return
330   -           jsr chrout      ;ausgeben
340   -hole       ldy #$00        ;offset auf null
350   -           lda (help),y    ;erstes namenszeichen holen
360   -           php             ;status merken
370   -           and #$7f        ;loeschen von bit 7
380   -           jsr chrout      ;zeichen ausgeben
390   -           iny
400   -           lda (help),y    ;zweites namenszeichen holen
410   -           php             ;wieder status merken
420   -           and #$7f        ;und bit 7 loeschen
430   -           bne ausg        ;2.zeichen existiert
440   -           lda #$20        ;leerzeichen
450   -ausg       jsr chrout      ;ausgeben
460   -           plp             ;2.status zurueckholen
470   -           bmi test        ;integer oder string
480   -;sonst funktion oder fliesskomma
490   -           plp             ;1.status zurueckholen
500   -           bmi funktion    ;funktion liegt vor
510   -           jmp float       ;fliesskommavariable liegt vor
520   -funktion   lda #$21        ;ascii fuer !
530   -           jsr chrout      ;ausgeben
540   -           jmp rest
550   -test       plp             ;1.status zurueckholen
560   -           bmi integer     ;beide bit 7 gesetzt = integervariable
570   -;stringvariable liegt vor
580   -string     lda #$24        ;$-zeichen
590   -           jsr chrout      ;ausgeben
600   -           lda #$20        ;leerzeichen
610   -           jsr chrout
620   -           lda #$3d        ;=-zeichen
630   -           jsr chrout
640   -           lda #$20        ;leerzeichen
650   -           jsr chrout
660   -           iny             ;offset auf stringlaenge richten
670   -           lda (help),y    ;laenge laden
680   -           tax             ;und merken
690   -           iny             ;offset auf stringadresse richten
700   -           lda (help),y    ;lsb adresse
710   -           sta index
720   -           iny
730   -           lda (help),y    ;msb adresse
740   -           sta index+1
750   -           jsr outstr      ;string ausgeben
760   -           jmp rest
770   -;
780   -integer    lda #$25        ;%-zeichen
790   -           jsr chrout      ;ausgeben
800   -           lda #$20        ;leerzeichen
810   -           jsr chrout
820   -           lda #$3d        ;=-zeichen
830   -           jsr chrout
840   -           lda #$20
850   -           jsr chrout
860   -           iny             ;offset auf msb richten
870   -           lda (help),y    ;msb laden
880   -           pha             ;und merken
890   -           iny
900   -           lda (help),y    ;lsb laden
910   -           tax             ;und merken
920   -           pla             ;msb in akku
930   -           jsr linprt      ;integerzahl ausgeben
940   -           jmp rest
950   -;
960   -float      lda #$20        ;leerzeichen
970   -           jsr chrout      ;ausgeben
980   -           lda #$20        ;noch ein leerzeichen
990   -           jsr chrout
1000  -           lda #$3d        ;=-zeichen
1010  -           jsr chrout
1020  -           lda #$20        ;und noch ein leerzeichen
1030  -           jsr chrout
1040  -           clc             ;addition vorbereiten
1050  -           iny             ;offset auf erstes wertebyte richten
1060  -           tya             ;und in akku schieben
1070  -           adc help        ;ergibt lsb
1080  -           pha             ;merken
1090  -           lda help+1      ;msb
1100  -           adc #$00        ;eventuell carry addieren
1110  -           tay             ;msb merken
1120  -           pla             ;lsb zurueckholen
1130  -           jsr movfm       ;fac mit variablenwert laden
1140  -           jsr numdon      ;fac ausgeben
1150  -rest       lda #$0d        ;carriage return
1160  -           jsr chrout
1170  -           clc             ;addition vorbereiten
1180  -           lda help        ;lsb
1190  -           adc #$07        ;auf naechste variable
1200  -           sta help
1210  -           lda help+1      ;msb
1220  -           adc #$00        ;eventuell carry addieren
1230  -           sta help+1
1240  -;
1250  -           lda help        ;vergleich, ob ende
1260  -           cmp arytab      ;der variablentabelle
1270  -           lda help+1
1280  -           sbc arytab+1
1290  -           bcs ende
1300  -           jmp hole        ;naechste variable
1310  -ende       rts
1320  -;
```

Listing 35. Hypra-Ass-Quelltext von DUMP C 64

Variablen auch dem Assemblerprogrammierer nützliche Dienste leisten.

Das gilt vor allem dann, wenn die Arrays im Basic-Rahmenprogramm eingerichtet und aus den Assemblerroutinen heraus angesteuert werden. Mehr noch als für die einfachen Variablen, deren Organisation wir in der letzten Folge kennengelernt haben, gilt es bei den indizierten Variablen, den Aufbau dieser Tabellen zu untersuchen, denn er ist wesentlich komplexer. Auch ist der Zugriff auf die einzelnen Elemente nicht so einfach zu realisieren. Wie also sind Arrays konstruiert? Wie kann man aus der Assemblerebene an die Elemente gelangen?

Der Kopf des Arrays

Allen Arrays ist ein sogenannter Header (oder Kopf) gemeinsam. Bei den unterschiedlichen Feldtypen (darauf gehen wir später noch ein) unterscheiden sich lediglich die beiden ersten Byte dieses Kopfes, die den Namen des Feldes und eine Typkennung enthalten, welche mit den Typkennungen bei den einfachen Variablen identisch ist. Zur Erinnerung:

Bit 7 in	Byte 1	Byte 2
Integer-Kennung:	1	1
Fließkomma-Kennung:	0	0
String-Kennung:	0	1

Bild 38 zeigt Ihnen den grundlegenden Aufbau des Arraykopfes. Im einfachsten Fall ist so ein Header sieben Byte lang. Auf die beiden ersten Byte (Name und Kennung) folgt eine Längenangabe: In zwei Byte wird die Gesamtlänge des Arrays (inklusive Kopf) in der Form LSB/MSB festgehalten.

Arrays dimensionieren

Theoretisch könnte ein Feld also 65535 Byte lang werden. Das folgende fünfte Byte im Header gibt die Anzahl der Dimensionen an: Es wären somit - wiederum theoretisch, denn wer kann das noch überschauen! - bis zu 255 Dimensionen möglich. In jeweils zwei Byte - und zwar im etwas ungewöhnlichen Format MSB/LSB - finden wir danach Angaben über die Elementanzahl je einer Dimension.

Zwei Dinge sind dabei noch zu beachten: Zum einen findet man hier immer eine um 1 höhere Elementanzahl als in der Dimensionierung. Ein Null-Element (in Programmiererkreisen gilt es als schick, nicht bei 1, sondern bei 0 mit dem Zählen anzufangen) wird mitgerechnet. Beispielsweise ergibt DIM A(4) hier die Zahl 5 (eben wegen der Reihenfolge A(0), A(1), A(2), A(3) und A(4)). Zum anderen aber gilt bei mehreren Dimensionen, daß für jede der genannten Dimensionen diese zwei Byte erscheinen, beginnend mit der zuletzt genannten Dimension. So ergibt beispielsweise DIM A(1,2,3) die folgende Belegung im Header vom 6.Byte an:

Byte 6: 0
Byte 7: 4 letzte genannte Dimension plus 1
Byte 8: 0
Byte 9: 3 vorletzte Dimension plus 1
Byte 10: 0
Byte 11: 2 erste Dimension plus 1

Die Länge des Kopfes ist also abhängig von der Anzahl der Dimensionen. Sie beträgt - wenn N diese Dimensionsanzahl symbolisiert - insgesamt:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Erster	Zweiter	LSB	MSB	Anzahl der	MSB	LSB
Buchstabe des Namens mit Kennung		der Arraylaenge (inclusive Kopf)		Dimensionen	der Anzahl an Elementen der letzten genannten Dimension	

Bild 38. Dies ist ein Arraykopf (an das siebte Byte schließen sich weitere Elementanzahlen an)

Bild 39. So sieht ein Element eines Integerfeldes aus...

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Exponent plus 128	Bit 7 ist fuer das Vorzeichen			
	Mantisse			

Bild 40. ...und so ein Element eines Fließkommafeldes. Bit 7 des zweiten Bytes dient als Vorzeichenkennung.

Byte 1	Byte 2	Byte 3
Stringlaenge	LSB	MSB
	des Stringtextortes	

Bild 41. Auf diese Weise ist ein Stringfeldelement aufgebaut. Es ist insgesamt drei Byte lang.

Länge = 5 + 2*N

Zu jedem Kopf gehört auch ein Körper; hier sind das die Elemente des Arrays, die sich nahtlos anschließen.

Die Array-Elemente

Im Gegensatz zu den einfachen Variablen - dort belegt jede, gleich welchen Typs, immer sieben Byte - verbrauchen die Array-Elemente unterschiedlich viel Speicherplatz. Sehen wir uns zunächst das Element eines Integer-Arrays an: Der Zahlenwert ist hier im Zwei-Byte-Format festgehalten. Allerdings findet man auch hier wieder die ungewöhnliche Reihenfolge MSB/LSB. Bild 39 zeigt solch ein Integer-Element. Ein Element eines Fließkomma-Arrays ist im üblichen MFLPT-Format angeordnet. Fünf Byte werden hier für eine Fließkommazahl benötigt, von denen das erste Byte dem Exponenten, die anderen vier der Mantisse zugeordnet sind. Das Bit 7 des zweiten Byte (also des ersten Mantissenbytes) dient als Vorzeichenkennung. Bild 40 zeigt Ihnen solch ein Fließkomma-Element.

Die Berechnung

Bleibt noch das String-Array-Element (Funktionen-Arrays – wie es die Einlagerung der Funktionen in die einfachen Variablen vermuten lassen würde – gibt es nicht). Solch ein String-Element besteht aus dem Stringdescriptor: Es ist daher drei Byte lang. Byte 1 gibt die Stringlänge, die Byte 2 und 3 den Stringtextort im Format LSB/MSB an. In Bild 41 finden Sie ein String-Array-Element.

Der Stringtext ist ebenso angeordnet wie bei den normalen Stringvariablen: Von der oberen Grenze des Basic-RAM abwärts. Auch hier findet sich die C 128-Besonderheit, daß im Anschluß an den Stringtext ein Zeiger auf den Stringdescriptor lokalisiert ist, der die Garbage-Collection beschleunigt (mehr darüber konnten Sie unter »Variablentypen des Basic« nachlesen).

Jetzt können wir auch genau den Speicherplatzbedarf eines Feldes errechnen. Wenn – wie oben – N die Anzahl der genannten Dimensionen symbolisiert und D_N, D_{N-1}, ..., D_1 die Längen der einzelnen Dimensionen (also letzte Dimension mit der Nummer N, vorletzte mit Nummer N-1, und so fort bis zur ersten Dimension mit der Nummer 1) sowie m den Platzbedarf eines Elementes (also m=2 für ein Integer-Element, m=5 für ein Fließkomma- und m=3 für ein String-Element) angibt, dann ergibt sich für die Länge des gesamten Array:

$$\text{Länge} = 5 + 2 * N + (D_N + 1) * (D_{N-1} + 1) * ... * (D_1 + 1) * m$$

An einem Beispiel soll Ihnen das deutlich werden. Nehmen wir ein Fließkomma-Feld A(12,20,45), dann gilt:

$$\text{Länge} = 5 + 2 * 3 + (45 + 1) * (20 + 1) * (12 + 1) * 5 = 62801$$

Hätten Sie das gedacht, daß solch ein Feld mit seinen 62801 Byte mit Sicherheit den Speicher sprengt? Als Integer-Array hätte es übrigens nur 25127 Byte verbraucht. So manchen Out of Memory Error kann man sich ersparen, wenn man den Platzbedarf vor dem Programmlauf berechnet.

Wo befindet sich ein bestimmtes Element?

Um nun auf ein bestimmtes Element eines Arrays zugreifen zu können, muß man natürlich wissen, wo es sich befindet. Relativ einfach verhält sich das bei einem eindimensionalen Feld. Nehmen wir an, wir hätten durch DIM A(5) ein eindimensionales Fließkommazahlen-Array definiert und es dann mit Inhalt versehen. Ein Blick mittels eines Monitors in den Basicspeicher zeigt, daß die Elemente in der Reihenfolge A(0), A(1), A(2), A(3), A(4), A(5) angeordnet sind. Um also das Element A(i) zu finden, muß man das (i+1)ste Element ansteuern. Das erste Byte unseres Elementes ergibt sich aus der Addition von 7 (Länge des Headers) und i*5 (fünf Byte bilden ein Fließkomma-Element) zum Start des Arraykopfes.

Sehr viel komplexer wird das Wiederfinden eines Elementes schon, wenn wir ein zweidimensionales Feld betrachten. Nehmen wir an, ein String-Array wäre durch DIM A$(2,3) definiert und dann mit Elementen versehen worden, dann ist die übliche Darstellung (als 2,3-Matrix) in Bild 42 zu sehen.

Ebenfalls darin eingezeichnet ist die Reihenfolge, die man im Anschluß an den Header mittels eines Monitors beobachten kann:

A$(0,0); A$(1,0); A$(2,0); A$(0,1); A$(1,1); A$(2,1);...;A$(2,3)

Die Speicherung findet also Spalte für Spalte statt. Arbeiten wir mit den vorhin schon verwendeten Bezeichnungen (D_1 für die Elementanzahl in der ersten genannten Dimension und so fort), dann können wir ganz allgemein eine Formel für den Ort eines Elementes A$(i,j) bei vorher durch DIM A$($D_1$, D_2) dimensionierten Feldern angeben. Die Nummer n des Elementes A$(i,j) ist dann nämlich:

$$n = (D_1 + 1) * j + i + 1$$

Beispielsweise steht dann das Element A$(2,1) aus dem obigen Array (das wir durch DIM A$(2,3) definiert hatten) an der Stelle

$$n = (2 + 1) * 1 + 2 + 1 = 6$$

Dies können Sie schnell nachprüfen: Das sechste Element ist tatsächlich A$(2,1). Das erste Byte eines beliebigen Elementes mit den Indices i,j und der Elementlänge m (wie vorhin schon gehabt: m=2 beim Integer-, 5 beim Fließkomma- und 3 beim String-Array) ergibt sich aus der Formel:

$$\text{Adresse 1. Byte} = \text{Headerstart} + 9 + m * ((D_1 + 1) * j + i)$$

Sie sehen: Es bedarf schon einiger Rechnereien, wenn man ein bestimmtes Element ansteuern möchte.

Die dritte Dimension

Durchaus nicht selten werden dreidimensionale Felder verwendet. Die Ansteuerung eines beliebigen Elementes ist hier noch etwas komplizierter. Gehen wir wieder von einem Beispiel aus: Durch DIM A(1,2,3) sei ein solches Array definiert worden und dann mit Fließkommazahlen gefüllt. Man kann es sich als einen Quader vorstellen, der die Tiefe von zwei hat (das entspräche der ersten genannten Dimension und betrifft die Indices (0,...,...) und (1,...,...)), die Höhe 3 (nämlich aus der zweiten Dimension mit den Indices (...,0,...), (...,1,...) und (...,2,...)) und die Breite 4 (hier ist es dann die dritte Dimension und die Indices (...,...,0), (...,...,1), (...,...,2) und (...,...,3)). Bild 43 zeigt Ihnen diese Gedankenstütze. Ebenfalls eingezeichnet als Linie ist die Reihenfolge der Elemente, die man mittels eines Monitors hinter dem Header entdeckt:

A(0,0,0); A(1,0,0); A(0,1,0); A(1,1,0); A(0,2,0); A(1,2,0); A(0,0,1); etc.

Es schält sich – wenn man die Elementanordnung durch die Dimensionen verfolgt – eine gewisse Gesetzmäßigkeit heraus: Offenbar wird die erste Dimension am häufigsten, die letzte am seltensten variiert. So findet man in den Elemente-Indices des obigen Beispiels die erste Dimension jedes zweite Mal, die zweite Dimension jedes dritte Mal und die dritte Dimension jedes siebte Mal variiert.

Die Nummer n eines Elementes A(i,j,k) nach dem Header eines Arrays, das durch DIM A (D_1,D_2,D_3) definiert und dann belegt worden ist, ergibt sich aus folgender Formel:

$$n = (D_1 + 1) * (D_2 + 1) * k + (D_1 + 1) * j + i + 1$$

Ein Beispiel (bezogen auf DIM A(1,2,3)): Die Nummer n des Elementes A(1,1,2) berechnet sich so:

$$n = 2 * 3 * 2 + 2 * 1 + 1 + 1 = 16$$

Das 16. Element heißt also A(1,1,2). Zählen Sie nach: Es stimmt! Damit ist es möglich, auch die Adresse des ersten Byte eines Elementes A(i,j,k) eines Feldes (das durch DIM A(D_1,D_2,D_3 definiert wurde) zu berechnen (m ist wieder die Länge eines Elementes):

$$\text{Adresse} = \text{Header-Startadresse} + 11 + m * ((D_1 + 1) * (D_2 + 1) * k + (D_1 + 1) * j + i)$$

Höhere Dimensionen als 3 werden schon recht selten sein. Trotzdem: Es zeichnet sich – wenn man sich die Formeln für n der Reihe nach ansieht – eine gewisse Systematik ab. So liegt es nahe, daß die Nummer n eines Elementes A(i,j,k,l) des 4dimensionalen Feldes so zu berechnen ist:

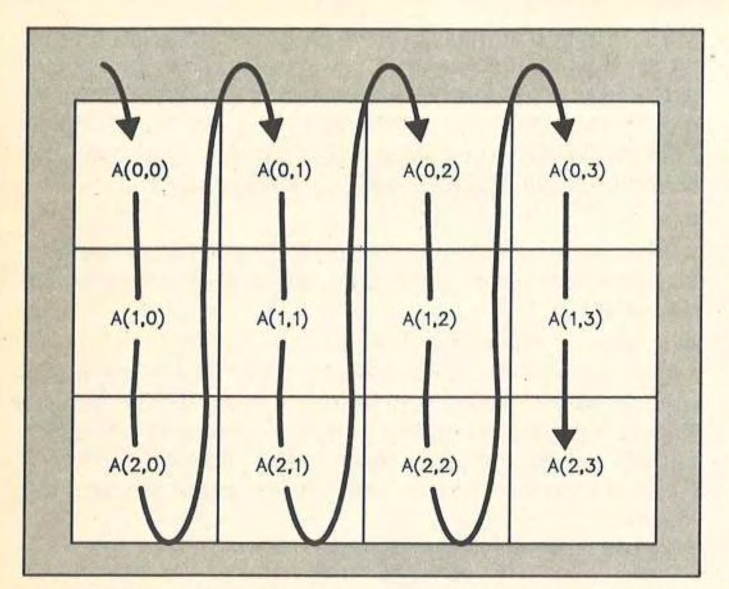


Bild 42. Die Elemente eines zweidimensionalen Feldes als (3,4)-Matrix und ihre Reihenfolge in der Feldtabelle

$$n = (D_1 + 1) * (D_2 + 1) * (D_3 + 1) * l + (D_1 + 1) * (D_2 + 1) * k + (D_1 + 1) * j + i + 1$$

Wenn Sie Lust dazu haben, dann probieren Sie diese Formel doch einmal aus: Mit der Pointer-Funktion des C128-Basic läßt sich das gut überprüfen.

Es gibt zwar eine Reihe von Basicinterpreter-Routinen, die speziell für den Umgang mit Arrays entwickelt wurden, sie sind aber meist nur sehr ungünstig über ein Assemblerprogramm anzusteuern.

Ansteuern der Feld-Elemente

So übernehmen diese Routinen die Angaben aus dem Basic-Test und schieben anschließend die Parameter in die verschiedensten Speicherstellen und auf den Stapel.

Oder, und dies ist eher noch komplizierter, die Routinen erfordern eine Unmenge teils sehr umständlicher Vorbereitungen. Parameter müssen hier an bestimmten Speicherstellen vorgegeben werden, eine Methode, die nicht nur Zeit kostet, sondern auch sehr fehleranfällig ist.

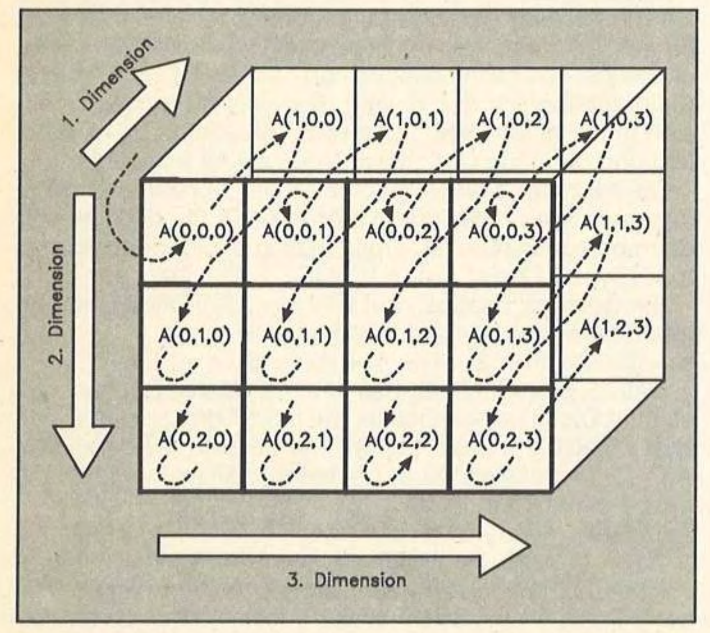


Bild 43. Die Elemente eines dreidimensionalen Feldes als (2,3,4)-Matrix. Die Reihenfolge ist durch die gestrichelten Pfeile angedeutet.

Es erscheint in fast jedem Fall sinnvoller, eine den eigenen Bedürfnissen angepaßte Routine selbst zu entwickeln, welche dann auch wesentlich komfortabler ausfallen kann als die vom Basic-Interpreter zur Verfügung gestellten.

Um Sie bei der Entwicklung einer solchen Routine zu unterstützen, möchten wir die Interpreter-Routinen an dieser Stelle abschließend kurz auflisten:

ISARY holt die Arrayparameter aus dem Basic-Text und legt sie auf den Stapel. Sie finden diese Routine beim C64 ab $B1D1, beim C 128 ab $7CAB.

FNDARY sucht in der Arraytabelle nach dem Namen. Die Routine steht beim C64 ab $B218, beim C 128 ab $7CF4. Der Name ist beim C 64 in $45/46, beim C 128 in $47/48 enthalten.

NOTFDD richtet ein neues Feld ein, wenn der Arrayname nicht gefunden wurde. Die Adressen der Routine: $B261 (C 64), $7D46 (C 128).

INLPN2 (C 64: $B30E, C 128: $7E00) sucht ein angegebenes Element und richtet den Zeiger VARPNT darauf. Diesen Zeiger finden Sie beim C 64 in $47/48, beim C 128 in $49/4A).

Eine sehr nützliche Routine ist aber **UMULTD**, eine 16-Bit-Multiplikationsroutine, die beim C 64 ab $B357 und beim C 128 ab $7E4B steht. UMULTD multipliziert eine Zahl in $28/29 (C 64) oder $72/73 (C 128). Das Ergebnis finden Sie in X/Y.

Entwickeln Sie einen Array-DUMP

Ein DUMP-Programm für die einfachen Variablen hatten wir bereits kennengelernt. Wir möchten Ihnen zum Abschluß noch eine Aufgabe stellen, die Sie mit den bisher erworbenen Assembler-Kenntnissen selbst lösen können.

Wie wäre es, wenn Sie ein ähnliches Programm wie DUMP entwickeln würden, das die Felder und deren Inhalte auf dem Bildschirm ausgibt? Zu Ihrer Unterstützung hier einige Wegmarken, die Ihnen bei der Lösung des Problems sicherlich weiterhelfen:

Der Basicspeicher von der Adresse an, auf die ARYTAB zeigt (das ist der Vektor $2F/30 (beim C 64) oder $31/2 (beim C 128), der den Beginn der Arraytabellen markiert), bis zu der Adresse, auf die STREND weist (dieser Zeiger kennzeichnet das Ende der Arraytabellen und liegt bei $31/2 (C64) oder $33/4 (C 128)) wird nach Arrayheadern durchforstet. Jeder gefundene Kopf ist dann zu untersuchen auf Name, Typ, Dimensionszahl und die Längen. Nun haben Sie mehrere Möglichkeiten: Sie können den Ausdruck der definierten Arrays veranlassen. Sie können aber auch nach dieser ersten Option den Inhalt eines ausgesuchten Feldes ausdrucken oder aber die Ausgabe sämtlicher Arrayinhalte ermöglichen. Für die beiden zuletzt genannten Optionen dürfte das DUMP-Programm aus der letzten Folge einige Hilfsmittel bieten: Untersuchung des Typs und danach gesteuerte Bildschirmausgaben. Haben Sie dann eine funktionierende Problemlösung gefunden, wäre es interessant, diese im 64'er-Magazin wiederzufinden, denn solche Utilities sind noch recht rar. Die Lösung für den C 128 dürfte wegen der Bank-Probleme erheblich schwieriger zu finden sein als für den C 64.

Dieser Kurs ist hiermit beendet. Wir hoffen, Sie sind nun selbst in der Lage, auch schwierige Problemlösungen durchzuführen. (Heimo Ponnath/C. Q. Spitzner

Keine Angst vor Maschinensprache

Wenn Sie bisher glaubten, Maschinensprache sei für Sie zu kompliziert und nur etwas für Profis, dann wird Ihnen dieser Assembler-Kurs zeigen, daß sich viele Dinge fast genauso einfach programmieren lassen wie in Basic – dabei wird die Programmausführung jedoch erheblich beschleunigt.

Die Vorgänge innerhalb des Computers beim Abarbeiten eines Programms sind relativ kompliziert. Eine solche Maschine »versteht« längst nicht alles, was man ihr mitteilen will. Tatsächlich kann sie nur zwischen zwei Zuständen unterscheiden, »Strom an« und »Strom aus«. Um diese beiden Zustände zu verdeutlichen, kennzeichnet man den ersten als 1, den zweiten als 0.

Wir können uns das etwa so vorstellen, daß jede Speicherstelle des Computers ein Haus mit acht Fenstern ist, von denen eine bestimmte Anzahl in einer gewissen Reihenfolge erleuchtet ist. In den erleuchteten Fenstern, beziehungsweise den Räumen dahinter, ist also der Strom an (=1), in den übrigen ist der Strom aus (=0). Ein Beispiel dazu sehen Sie in Bild 1.

Ein Bus für 8 und 16 Bit

Über den acht Leitungen breiten Datenbus werden diese Informationen nun an den Prozessor geliefert, der entsprechend der 1/0-Kombination seine Operationen ausführt. Zusätzlich muß der Prozessor wissen, was er genau zu tun hat; er benötigt einen Befehl. Ein solcher Befehl besteht ebenfalls wieder aus einer 1/0-Kombination.

Während also einerseits die Zahlenkombination 01001100 (binär) die Zahl 76 (dezimal) als Inhalt einer Speicherstelle beschreibt, bedeutet diese Kombination andererseits den Assemblerbefehl JMP, gleichzusetzen mit dem GOTO-Befehl in Basic.

Der Prozessor muß nun wissen, ob er die Kombination als Wert oder als Befehl verstehen soll. Deshalb bestimmt der jeweils letzte Befehl, ob ein Wert folgt. Der C 64 beispielsweise verfügt über einen Acht-Bit-Datenbus, das heißt, daß jeweils acht Bit (= 1 Byte) gleichzeitig übertragen werden können. Gleichzeitig kann auch in jeder Speicherstelle maximal ein Byte gespeichert werden. Daher wird bei Befehlen, die mit solchen Werten operieren, grundsätzlich das nächste Byte als Zahl interpretiert.

Low- und High-Byte

Neben dem Datenbus gibt es im Computer noch einen Adreßbus, der eine Breite von 16 Bit hat. Über diesen Adreßbus werden die einzelnen Speicherstellen angesprochen. Während acht Bit insgesamt $2^8 = 256$ verschiedene Kombinationen zulassen, kann man über einen 16-Bit-Bus insgesamt $2^{16} = 65536$ verschiedene Speicherstellen adressieren. Diese hohen Zahlen müssen in einem 8-Bit-Computer zur Verarbeitung in zwei einzelne Byte, das Low-Byte und das High-Byte, aufgespalten werden. Deshalb erwartet der Prozessor nach einem Befehl, der sich auf eine Adresse bezieht, zum Beispiel dem Befehl JMP, einen Zwei-Byte-Wert in der Reihenfolge Low-Byte, High-Byte.

Außerdem gibt es noch Befehle, die keine zusätzlichen Daten erfordern. Erhält der Computer einen solchen Befehl, führt er ihn aus und erwartet sofort den nächsten. Entscheidend ist jedoch bei allen drei Versionen, daß der Prozessor ausschließlich Bitmuster erkennt und verarbeiten kann.

Es ist leicht, sich vorzustellen, welcher Aufwand computerintern getrieben werden muß, um beispielsweise eine Basic-Zeile in das 1/0-Format umzuwandeln, so daß sie ausgeführt werden kann. Trotz der großen Geschwindigkeit, mit der der Prozessor einzelne Byte-Befehle ausführt, macht sich bei großen Programmen oder aufwendigen Berechnungen der Zeitverlust bei der »Übersetzung« des Basic-Programms höchst negativ bemerkbar.

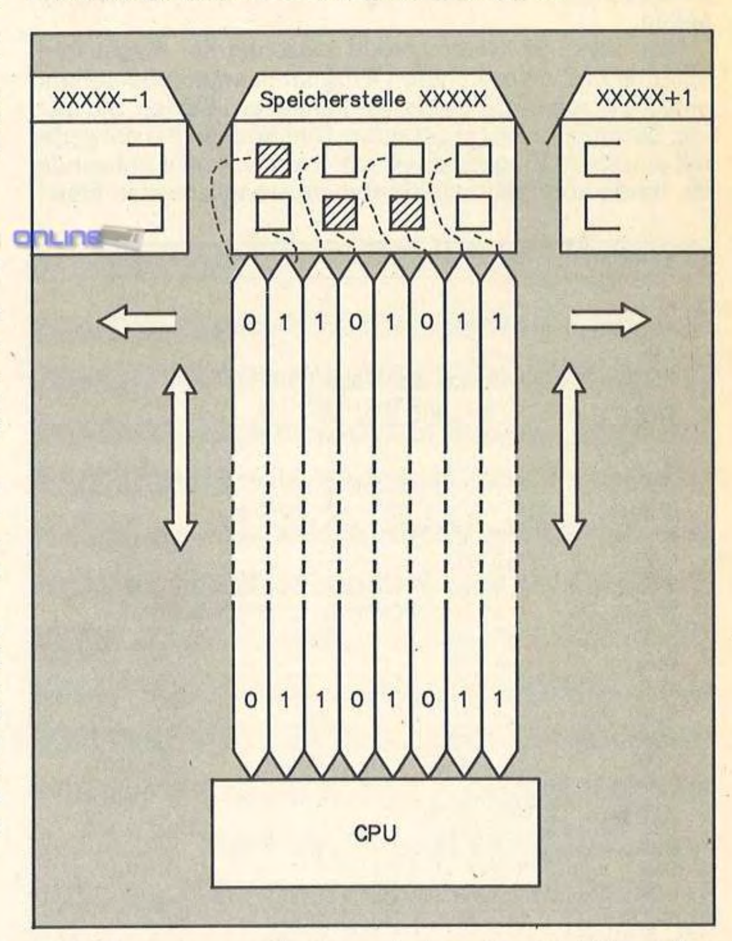


Bild 1. Datenkommunikation zwischen einer Speicherstelle und dem Prozessor (CPU)

Es wäre also wünschenswert, dem Computer das Programm gleich in maschinenlesbarer Form mitzuteilen. Es bieten sich dazu in erster Linie zwei Verfahrensweisen an:

Einerseits kann man einen Compiler einsetzen. Ein Compiler wandelt das fertige Programm, geschrieben in einer höheren Programmiersprache wie etwa Basic, in ein maschinenlesbares Programm, einen sogenannten

Objektcode, um. Das ursprüngliche Programm, der Quellcode, wird danach nicht mehr benötigt, ein reines Maschinensprache-Programm wirkt nun im Computer.

Assembler: Stein auf Stein

Der Zeitgewinn gegenüber dem Basic-Interpreter ist dabei schon sehr groß. Allerdings hat dieses Verfahren einen Nachteil. Bei der Umsetzung wird nicht immer der beste Weg beschritten, so daß das Maschinenprogramm länger wird, als nötig wäre. Dies kommt daher, daß Compiler dem erzeugten Code zusätzlich noch ein sogenanntes »Runtime-Modul« anfügen. Es wird also wertvoller Speicherplatz verschwendet und auch zeitmäßig ist das Optimale noch nicht erreicht.

Andererseits kann man mit Hilfe eines Assemblers oder Maschinensprache-Monitors direkt in Maschinensprache programmieren und erhält so (bei entsprechend geschickter Programmierung) die kürzeste und schnellste Version. Allerdings ist das Programmieren in Maschinensprache nicht ganz einfach. So ist für den Anfänger zunächst die große Anzahl verschiedenartiger Befehle ein Problem. Schwierigkeiten bereitet aber auch die Einfachheit der Anweisungen, aus denen große Programme entstehen sollen.

Man kann die Unterschiede zwischen der Maschinensprache und einer höheren Programmiersprache anhand eines einfachen Beispiels verdeutlichen: Stellen Sie sich vor, Sie müßten ein Haus bauen. Dies können Sie entweder mit einzelnen Ziegelsteinen tun, was zwar sehr aufwendig ist, Ihnen aber optimale Gestaltungsmöglichkeiten bietet. Ein schnellerer Weg zum eigenen Heim ist die Verwendung von Fertigbauteilen, wobei Sie aber hinsichtlich der Feinheiten der Gestalung einige Abstriche machen müssen. Diese vorgefertigten Teile symbolisieren die Befehle einer höheren Programmiersprache. Viele »kleine« Assemblerbefehle sind hier zu einem Klotz zusammengefaßt, der zwar schneller zum Ziel führt, dafür aber nicht mehr so universell eingesetzt werden kann.

Das Beispiel mag etwas hinken, das Prinzip gilt aber dennoch: Durch Assembler-Programme kann man Probleme gezielter, schneller und mit weniger Aufwand lösen, die Programmierung selbst ist aber etwas kompliziert. Bei der Assembler-Programmierung verzichtet man auf die Eingabe von Einsen und Nullen und verwendet statt dessen leichter zu merkende Kürzel für die Befehle, sogenannte »Mnemonics«. Das Wort kommt von griechisch »Mneme«, dem Begriff für Gedächtnis. Es ist ein leichtes, Analogien zwischen den Assembler- und Basic-Befehlen zu entdecken. Ein Beispiel:
Assembler: JMP Basic: GOTO

Mnemonics: Kürzel für einfaches Programmieren

Weitere Beispiele finden Sie in Tabelle 1. Das Verständnis der Mnemonics wird dadurch zu Beginn Ihrer Maschinensprache-Karriere erleichtert. Es ist allerdings zu beachten, daß die Vergleiche nur mit Einschränkungen gültig sind, beziehungsweise die Bedeutung der Assemblerbefehle nur unvollständig wiedergeben.

Assembler:	Basic:
ADC	--
AND ...	A AND ...
ASL	--
BCC M	IF W <= 255 THEN GOTO M
BCS M	IF W > 255 THEN GOTO M
BEQ M	IF W1 = W2 THEN GOTO M
BIT M	--
BMI M	IF W < 0 THEN GOTO M
BNE M	IF W1 <> W2 THEN GOTO M
BPL M	IF W > 0 THEN GOTO M
BRK	(STOP)
BVC M	IF W > -127 OR W < 127 THEN GOTO M
BVS M	IF W < -127 OR W > 127 THEN GOTO M
CLC	--
CLD	--
CLI	--
CLV	--
CMP ...	... = A ?
CPX ...	... = X ?
CPY ...	... = Y ?
DEC M	M = M - 1
DEX	X = X - 1
DEY	Y = Y - 1
EOR ...	... EXOR A
INC M	M = M + 1
INX	X = X + 1
INY	Y = Y + 1
JMP M	GOTO M
JSR M	GOSUB M
LDA ...	A = ...
LDX ...	X = ...
LDY ...	Y = ...
LSR	--
NOP	Leerzeile
ORA ...	A OR ...

Assembler:	Basic
PHA	--
PHP	--
PLA	--
PLP	--
ROL	--
ROR	--
RTI	--
RTS	RETURN
SBC ...	(A = A - ...)
SEC	--
SED	--
SEI	--
STA M	POKE M, A
STX M	POKE M, X
STY M	POKE M, Y
TAX	X = A
TAY	Y = A
TSX	--
TXA	A = X
TXS	--
TYA	A = Y

Hierbei bedeuten:

A	= Inhalt des Akkumulators
X	= Inhalt des X-Registers
Y	= Inhalt des Y-Registers
M	= (Speicher-)Adresse
W	= Ergebnis vorhergehender Berechnungen beziehungsweise Zahl (Wert)
...	= Zahl (Wert) oder (Speicher-)Adresse
--	= zu diesem Befehl konnte keine sinnvolle Basic-Analogie gefunden werden

Basic-Analogien in Klammern sind nur bedingt richtig. Indirekte oder indizierte Adressierungen werden in der Tabelle nicht behandelt.

Tabelle 1. Analogien der Assembler-Befehle der 6510-CPU zu Basic

Sollten Sie keinen Assembler besitzen, empfiehlt es sich, unser Programm »Giga-Ass« aus diesem Sonderheft (Seite 116) abzutippen. Dies ist ein hervorragender Assembler, der Sie auch in die Lage versetzt, die Beispiele am Computer nachzuvollziehen oder eigene Maschinensprache-Programme zu schreiben.

Für den Anfänger bietet sich in erster Linie die Gelegenheit, einzelne Programmstücke mit dem Assembler zu schreiben, die man dann in eigene Spiele oder andere Programme einbaut. Als Beispiel wollen wir uns hier die Basic-Programmierung einer Joystick-Abfrage ansehen:

Angenommen, Sie wollen ein Sprite horizontal oder vertikal über den Bildschirm bewegen. Dabei beschränken wir uns auf die »normalen« 255 Bildschirmpositionen, das heißt, wir verzichten zunächst auf die X-Adressen 256 bis 320, die nur über das MSB (Most Significant Bit) zu erreichen sind. Wählen wir Sprite Nummer 0, so ist die Bildschirmposition in den Adressen 53248 (X-Position) und 53249 (Y-Position) abgelegt. Zur Steuerung benutzen wir den Joystickport 2 (Adresse 56320).

Zunächst definieren wir die Variablen:

```
100 A=PEEK(56320)
110 X=PEEK(53248)
120 Y=PEEK(53249)
```

Danach fragen wir eine eventuelle Joystickbewegung ab. Durch eine logische Und-Verküpfung wird geprüft, ob das entsprechende Bit gelöscht ist:

```
200 IF (A AND 1) = 0 THEN GOTO 300
210 IF (A AND 2) = 0 THEN GOTO 310
220 IF (A AND 4) = 0 THEN GOTO 320
230 IF (A AND 8) = 0 THEN GOTO 330
240 RETURN
```

Nun wird, falls erforderlich, die Position des Sprites verändert:

```
300 Y=Y-1
301 POKE53249,Y
302 RETURN
310 Y=Y+1
311 POKE53249,Y
312 RETURN
320 X=X-1
321 POKE53248,X
322 RETURN
330 X=X+1
331 POKE53248,X
332 RETURN
```

Diese Abfrage, als Unterprogramm geschrieben, kann beliebig oft aus dem Hauptprogramm durch »GOSUB 100« aufgerufen werden. Auf Sicherheitsabfragen wurde aus Gründen der Übersichtlichkeit verzichtet.

Die Umsetzung in ein Maschinensprache-Programm ist mit Hilfe unserer Tabelle 1 nun recht einfach:

```
100 LDA 56320; Lade Akkumulator mit dem Inhalt
    der Speicherstelle 56320.
    Absolute Adressierung
110 LDX 53248
120 LDY 53249
200 AND  #1
201 CMP  #0; Ist das Ergebnis 0?
202 BEQ MA1; Wenn ja, Sprung zu
    Marke 1. Relative Adressierung
203 LDA 56320; da der Inhalt des
    Akkumulators durch die Und-Verknüpfung
    geändert wurde
210 AND  #2
211 ...
```

Die folgenden Schritte verlaufen nach demselben Muster:

```
230 AND  #8
231 CMP  #0
232 BEQ MA4
240 RTS
300 MA1  DEY
301 STY 53249; Speichere den Inhalt des
    Y-Registers in 53249
302 RTS
310 MA2  INY
311 STY 53249
312 RTS
320 MA3  DEX
321 STX 53248
322 RTS
330 MA4  INX
331 STX 53248
332 RTS
```

Als Startadresse kann man beispielsweise 49152 festlegen:

```
010 .BASE 49152
```

Das Quellprogramm wird nun mit

```
400 .END
```

beendet. Nach dem Assemblieren liegt das Programm in reiner Maschinensprache vor. Wir starten die Joystick-Abfrage nun mit »SYS 49152«. Nach Erreichen eines »RTS«-Befehls kehrt das Programm zurück ins Basic. Wenn Sie diese Routine in ein Basic-Programm einbinden, werden Sie den Geschwindigkeitszuwachs gegenüber der Basic-Abfrage erkennen. Aber damit ist noch nicht die Grenze erreicht. Eine solche Abfrage ist durchaus effektiver, das heißt, noch kürzer programmierbar. Wir haben hier nämlich einen recht umständlichen Weg gewählt, um uns Ähnlichkeiten mit Basic zunutze zu machen, und so das Assembler-Programm besser zu verstehen.

Sicherheit vor dem Absturz

Bei praktischer Verwendung des Maschinenprogramms wird übrigens eine Sicherheitsabfrage für die Bereichsüberschreitung unverzichtbar, da beim Absturz eines Maschinensprache-Programms oft nur ein Reset beziehungsweise das Ausschalten des Computers die einzige Lösung bleibt. Bei der Sicherheitsabfrage kann man etwa nach folgendem Schema vorgehen:

```
111 CPX #255; ist X=255?
112 BEQ MA5; wenn ja, dann ...
113 JMP MA6
114 MA5 DEX      ; ...X um 1 vermindern
115 MA6 CPX #0  ; ist X=0?
116 BEQ MA7 ; wenn ja, dann ...
117 JMP MA8
118 MA7 INX      ; ...X um 1 erhöhen
120 MA8 ...
```

Diese Überprüfung muß selbstverständlich auch für den Y-Wert erfolgen. Die Joystickabfrage sollte dann ohne Schwierigkeiten funktionieren.

Die hochkomplizierten Vorgänge bei der Ausführung eines Programms spielen sich zum überwiegenden Teil in einem einzigen Register ab. Das heißt, alle Operationen laufen nacheinander ab, zerlegt in winzig kleine Teilschritte. Hier liegt auch der Grund dafür, daß bei einer Taktfrequnz von fast einer Million Hertz pro Sekunde (während eines Taktes wird eine Ein-Byte-Operation, entweder ein Assemblerbefehl oder ein Speicherzugriff, ausgeführt) dennoch Rechenzeiten von mehreren Sekunden auftreten.

Welche Register besitzt nun der 6510-Prozessor im C64 im einzelnen? Betrachten Sie dazu Bild 2. Da ist zunächst einmal das eigentliche Arbeitsregister des Prozessors, der Akkumulator. Der Akkumulator ist ein 8-Bit-Register, das heißt, in ihm findet genau ein Byte Platz. Die gleiche Größe besitzen das X- und das Y-Register, die sogenannten Indexregister. Sie dienen hauptsächlich zum Abarbeiten von Tabellen und sind nützlich für Transfer-Operationen sowie für Zählvorgänge. A, X und Y werden für die weitere Arbeit bestimmte Werte zugeordnet.

Ein Register, das insbesondere Funktionen für das Arbeiten mit Maschinensprache besitzt, ist das Status-Register. Im Status-Register befinden sich sogenannte Flags. Das sind Bits, deren jeweilige Zustände (1 oder 0) den Prozessor veranlassen, bestimmte Operationen auszuführen. Welche Flags es gibt und welche Kriterien ihren Zustand bestimmen, werden wir später noch behandeln.

Der Stack-Pointer verwaltet den Prozessorstack. Der Prozessorstack ist ein Zwischenspeicher, auf den bei Bedarf der Inhalt des Akkumulators und des Statusregisters gerettet wird, um Platz für andere Operationen zu schaffen. Der Prozessorstack arbeitet nach dem LIFO-Prinzip (Last In - First Out), der zuletzt auf dem Stack abgelegte Wert wird als erster wieder ausgelesen (Bild 3). Der Stack-Pointer zeigt immer auf diese Stelle.

Das letzte für die Programmierung wichtige Register ist der Programmzähler, ein 16-Bit-Register. Im Programmzähler befindet sich immer die Adresse der Programmstelle, die als nächstes zu bearbeiten ist. Bei jedem Takt wird dieses Register automatisch hochgezählt.

Flaggenparade

Das Status-Register beinhaltet insgesamt sieben verschiedene Flags, zu deutsch Flaggen. Sieben Bits können also, je nachdem ob sie gesetzt oder nicht gesetzt sind, eine besondere Bedeutung für den weiteren Ablauf des Programms haben. Die Anordnung der Bits im Register ist folgende:

N	V	-	B	D	I	Z	C

C: Das Carry-Flag wird gesetzt, wenn bei Additionen ein Übertrag auftritt. Im Akkumulator können beispielsweise aufgrund der Größe nur Ergebnisse bis 255 verarbeitet werden. Wird dieser Bereich nun bei einer Addition überschritten, wird das Carry-Flag gesetzt. Befehle SEC, CLC.

Z: Das Zero-Flag wird gesetzt, wenn das Ergebnis einer Operation gleich Null ist.

I: Das Interrupt-Disable-Flag wird gesetzt, wenn ein Interrupt nicht stattfinden darf, da eine Operation nicht gestört werden soll. Das ist zum Beispiel beim Verbiegen des IRQ-Vektors der Fall. Der Befehl SEI bewirkt dies. CLI erlaubt den Interrupt wieder.

D: Das Dezimal-Flag schaltet zwischen Binär- und Dezimalarithmetik um. Ist das Dezimal-Flag gesetzt, werden Zahlen im Akku als Dezimalzahlen interpretiert. Die unteren vier Bits werden bis neun hochgezählt, dann erfolgt ein Übertrag auf die oberen vier Bits, so daß der Wert des Registers insgesamt 99 betragen kann. Befehle: SED, CLD.

B: Das Break-Flag wird gesetzt, wenn der Break-Befehl ausgeführt wird.

V: Das Overflow-Flag ist ein Signal, daß bei einer vorzeichenbehafteten 8-Bit-Addition der zulässige Bereich überschritten wurde. Der zulässige Bereich erstreckt sich von -128 bis +127. Overflow-Flag löschen: CLV.

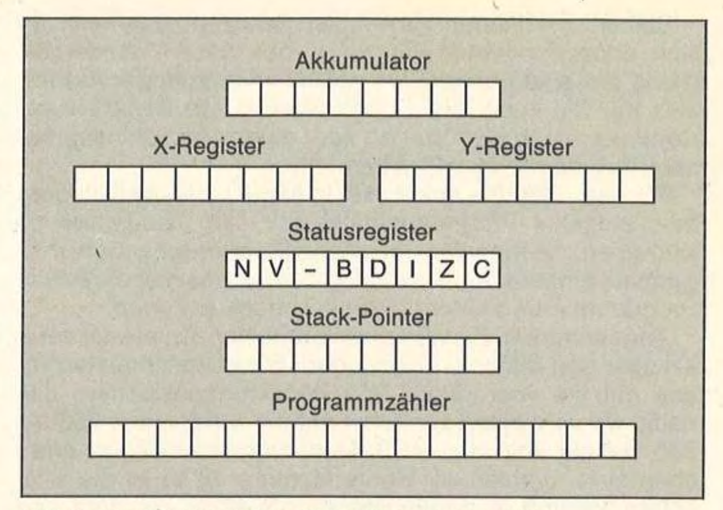


Bild 2. Die Register des 6510-Mikroprozessors

N: Das Negativ-Flag ist gesetzt, wenn das Ergebnis einer Operation negativ ist.

Um eine Vielzahl verschiedener Lade- und Speicheroperationen zu gewährleisten, gibt es mehrere Adressierungsarten. Die einfachste ist die unmittelbare Adressierung:

LDA #$FF

lädt den Akkumulator mit dem Wert 255 (hexadezimal $FF).

Eine weitere Adressierung ist die absolute Adressierung:

LDA $C000

lädt den Akkumulator mit dem Inhalt der Speicherstelle 49152 (hexadezimal $C000).

Komplizierter wird die Sache schon bei den indizierten Adressierungen:

LDA $C000,X

lädt den Akkumulator mit dem Inhalt der Speicherstelle (49152+Inhalt des X-Registers). Es handelt sich um eine X-indizierte Adressierung. Ebenso kann man auch Y-indiziert adressieren:

LDA $C000,Y

Durch die X- oder Y-Adressierung ist es leicht, Tabellen abzuarbeiten, indem man die Basisadresse beibehält und das X- oder Y-Register hochzählt.

Die Befehle benötigen mit Ausnahme der unmittelbaren Adressierung jeweils drei Byte Speicherplatz. Sie werden im Speicher folgendermaßen abgelegt:

LDA $C000: AD 00 CO

»AD« ist der hexadezimale Code für den absoluten LDA-Befehl (siehe auch Tabelle 2), »00 CO« ist die Adresse in der Reihenfolge Low-Byte, High-Byte. Um Speicherplatz und Rechenzeit zu sparen, kann man die oben erwähnten

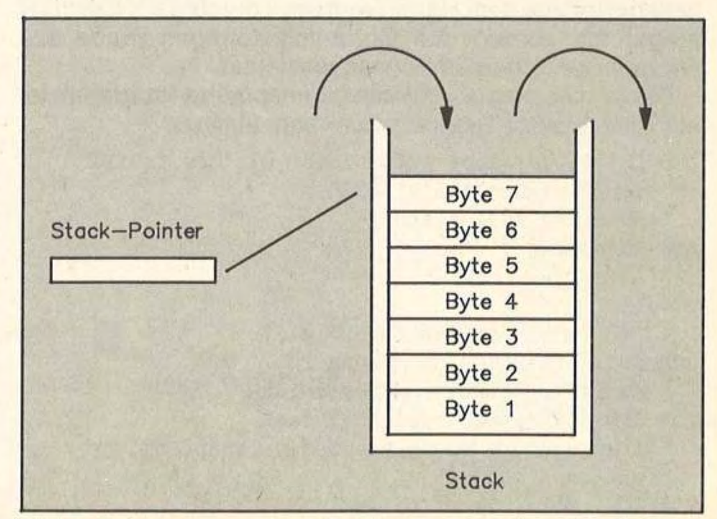


Bild 3. Der Prozessorstapel wird stets von oben bearbeitet

Adressierungen in der Zero-Page als 2-Byte-Befehle anwenden. Die Zero-Page, zu deutsch »Null-Seite«, ist der Speicherbereich von $00 bis $FF, also alle Adressen kleiner oder gleich 1 Byte:
LDA $5E : A5 5E
Es gibt darüber hinaus noch weitere Adressierungsarten, beispielsweise die indiziert-indirekte Adressierung. Diese Adressierung ist nur innerhalb der Zero-Page in Kombination mit dem X-Register erlaubt:
LDA ($5E,X)
Der Akkumulator wird daraufhin mit dem Inhalt der Speicherzelle geladen, die sich aus folgender Rechnung ergibt:
LDA $((5E+X)+(256*(5E+X+1)))
Der Inhalt des X-Registers wird also zu $5E addiert. In der daraus resultierenden Speicherzelle befindet sich das Low-Byte, in der nächsten Speicherzelle das High-Byte der endgültigen Adresse.
Zur Veranschaulichung hier ein Beispiel:
X = $10
Low-Byte = PEEK($5E+$10 = $6E)
High-Byte = PEEK($6F)
PEEK($6E) = $00
PEEK($6F) = $C0
Adresse = $C000
Das ist nicht ganz einfach zu berechnen, aber sehr nützlich innerhalb eines Programms.
Eine Adressierung, speziell zur Verwendung in Verbindung mit dem Y-Register, ist die indirekt-indizierte Adressierung. Die Ausführung ist ähnlich der indiziert-indirekten Adressierung:
LDA ($5E),Y entspricht: LDA (($5E+256*$5F)+Y)
Y = $10
Low-Byte = PEEK($5E)
High-Byte = PEEK($5F)
PEEK($5E) = $00
PEEK($5F) = $C0
Adresse = $C010
Daneben gibt es noch die sogenannte relative Adressierung. Diese Adressierung wird bei den bedingten Sprungbefehlen angewandt. Die bedingten Sprungbefehle, wie beispielsweise BCC oder BCS, erlauben nur Sprünge über Distanzen bis zu 128 Byte. Es wird keine direkte Sprungadresse, sondern die Differenz, der sogenannte »Offset«, zur Zieladresse im Speicher abgelegt. Dabei sind Werte von 1 bis 127 für Vorwärtssprünge und Werte von 128 bis 255 für Rücksprünge zuständig. Die Sprungadresse für Rücksprünge berechnet sich wie folgt:
Sprungziel = Programmzähler + Wert — 256
Der Programmierer hat in der Regel nichts mit der Berechnung des Offsets zu tun. Er kann die Zieladresse absolut angeben:
C000 BCC $C020
Überschreitet der Offset die zulässige Bytezahl, erhält er einen »Branch Error« vom Assembler gemeldet.
Eine weitere Adressierungsart, nämlich die indirekte Adressierung erlaubt der JMP-Befehl. Bei JMP (Adresse) wird die Adresse angesprungen, die in »Adresse« und »Adresse + 1« im Low-, Highbyte-Format abgelegt ist. Im C64 wird diese Adressierungsart für die Sprungtabelle im Kernel genutzt.
Eine letzte Adressierungsart bleibt zu erwähnen, die implizite Adressierung. Hier wird durch den Befehl selbst die Adresse bestimmt, das heißt, es ist nicht nötig, ein Sprungziel anzugeben. Beispiel:
RTS
Der Befehl »RTS« (Return from Subroutine) ist vergleichbar mit dem »RETURN«-Befehl in Basic und wird automatisch zur richtigen Adresse ausgeführt.

Die Prozessor-Befehle

Der 6510-Prozessor kennt eine Vielzahl von Befehlen, die einer kurzen Erklärung bedürfen. Wir stellen Ihnen deshalb die Befehle in einer alphabetischen Übersicht vor. Vergleichen Sie dazu auch die Tabellen 1 und 2.
ADC: »Add with Carry«. Der Akkumulatorinhalt, der Operand und das Carry-Bit werden addiert. Das Carry-Flag wird gesetzt, wenn das Ergebnis größer als 255 ist. Folgende Flags können durch den ADC-Befehl beeinflußt werden: N,V,Z,C
AND: »And Akku with Memory«. Akkumulator und Operand werden UND-verknüpft. Beeinflußte Flags: N,Z
ASL: »Arithmetic Shift Left«. Die Bits des Akkumulators oder des Operanden werden um eine Stelle nach links geschoben. Der so entstehende »Leerraum am rechten Ende« wird durch eine Null gefüllt, das »links herausrutschende Bit« wandert ins Carry-Flag. Der Vorgang entspricht einer Multiplikation mit 2. Flags: N,Z,C
BCC: »Branch if Carry Clear«. Verzweigt, wenn das Carry-Flag gelöscht ist.
BCS: »Branch if Carry Set«. Verzweigt, wenn das Carry-Flag gesetzt ist.
BEQ: »Branch if Equal«. Verzweigt, wenn das Zero-Flag gesetzt ist. Bei Vergleichsoperationen wird das Zero-Flag beeinflußt. Deshalb wird es häufig zur Prüfung der Gleichheit benutzt (Equal: gleich).
BIT: »Bit Test«. Das 6. und 7. Bit des angegebenen Operanden werden im Negativ- und Overflow-Flag abgelegt. Dann wird der Akkumulator mit dem Operanden logisch AND-verknüpft. Die Register werden bis auf die Flags nicht verändert. Flags: N,V,Z
BMI: »Branch if Minus«. Verzweigt, wenn das Ergebnis einer Rechnung negativ und somit das Negative-Flag gesetzt ist.
BNE: »Branch if Not Equal«. Gegenstück zu BEQ.
BPL: »Branch if Plus«. Gegenstück zu BMI.
BRK: »Break«. Der Befehl dient dazu, beim Austesten von Programmen sogenannte »Break-Points« zu setzen. Beim Erreichen eines solchen Punktes stoppt das Programm und verzweigt in eine Unterprogramm-Routine, meistens einen Maschinensprache-Monitor. Zuvor werden Programmzähler und Status auf den Stack gerettet. Dann springt das Programm zu der in $FFFE und $FFFF angegebenen Adresse. Die Rückkehr ist mit RTI zu veranlassen. Flags: B,I
BVC: »Branch if Overflow Clear«. Verzweigt, wenn das Overflow-Flag gelöscht ist.
BVS: »Branch if Overflow Set«. Gegenstück zu BVC.
BVS und BVC sind Befehle zur bedingten Verzweigung; die folgenden »Clear«-Befehle dienen dazu, das jeweilige Flag direkt zu beeinflussen.
CLC: »Clear Carry-Flag« löscht das Carry-Flag. Der Befehl sollte vor einer Addition verwendet werden, um zu verhindern, daß ein eventuell gesetztes Carry-Bit (Übertrag) addiert wird.
CLD: »Clear Decimal-Flag« löscht das Dezimal-Flag und schaltet so die Dezimal-Arithmetik ab.
CLI: »Clear Interrupt-Disable-Flag« löscht das Interrupt-Flag. Von nun an sind Unterbrechungen (Interrupts) erlaubt, die über den IRQ-Eingang angemeldet werden.
CLV: »Clear Overflow-Flag« löscht das Overflow-Flag.
CMP: »Compare to Accumulator« vergleicht einen Wert mit dem Inhalt des Akkumulators. Der Akkumulator wird nicht verändert. Die Operation läuft wie folgt ab: Der Vergleichswert wird vom Inhalt des Akkumulators abgezogen. Das Ergebnis wird nicht gespeichert, der Akkumulator bleibt

also unverändert. Die Rechenoperation hat jedoch die Flags beeinflußt. Anhand der gesetzten Flags kann man also gewisse Schlüsse ziehen. Sind beispielsweise Wert und Akkumulatorinhalt gleich, so ergibt die Rechnung A-W null. Daraufhin wird das Zeroflag gesetzt. Soll bei Gleichheit ein Unterprogramm angesprungen werden, benutzt man nun einfach den BEQ-Befehl.

Die Subtraktion beeinflußt darüber hinaus noch das Carry- und das Overflow-Flag. Das Carry-Flag wird gesetzt, wenn der Wert kleiner oder gleich dem Akkumulator war. Man kann also folgende Verzweigungstabelle aufstellen:

W < A:	BCS
W = A:	BEQ
W > A:	BCC
W < > A:	BNE

CPX: »Compare to X« wie CMP, allerdings wird anstatt Akkumulator das X-Register zum Vergleich herangezogen.
CPY: »Compare to Y« wie CMX, jedoch mit Y-Register
DEC: »Decrement Memory« vermindert den Wert der danach angegebenen Speicherzelle um 1, beeinflußt dadurch gegebenenfalls das N- und Z-Flag.
DEX: »Decrement X« vermindert den Inhalt des X-Registers um 1. Ideal zum Programmieren von Schleifen (ähnlich »FOR I = X TO 0 STEP -1«). Beeinflußt N- und Z-Flag.
DEY: »Decrement Y« vermindert den Inhalt des Y-Registers um 1.
EOR: »Exclusive-Or Memory with Accumulator« verknüpft den Inhalt des Akkumulators mit einem Wert Exklusiv-ODER. Im Akkumulator wird ein Bit genau dann gesetzt, wenn es entweder im Akkumulator oder im Speicher gesetzt war, nicht aber, wenn es im Akkumulator und im Speicher oder überhaupt nicht gesetzt war. Beeinflußt gegebenenfalls N-, Z- und C-Flag.
INC: »Increment Memory« erhöht den Inhalt der angegebenen Speicherstelle um 1 und beeinflußt gegebenenfalls N- und Z-Flag.
INX: »Increment X« erhöht den Inhalt des X-Registers um 1 und beeinflußt gegebenenfalls N- und Z-Flag.
INY: »Increment Y« erhöht den Inhalt des Y-Register um 1 und beeinflußt gegebenenfalls N- und Z-Flag.
JMP: »Jump to« führt einen unbedingten Sprung zur angegebenen Adresse aus, wobei die Adressierung sowohl absolut als auch indirekt erlaubt ist. Es werden keine Flags beeinflußt.
JSR: »Jump to Subroutine« führt einen Unterprogramm-Sprung aus. Der momentane Programmstand wird auf den Prozessor-Stack gerettet. Wenn das Unterprogramm durch RTS abgeschlossen wird, erfolgt ein Rücksprung zur Adresse direkt hinter dem JSR-Befehl. Hier werden ebenfalls keine Flags beeinflußt.
LDA: »Load Accumulator with« lädt den Akkumulator mit dem angegebenen Wert (unmittelbar) oder mit dem Inhalt der angegebenen Speicherstelle (absolut oder indiziert). Dem Wert entsprechend werden N- und Z-Flag verändert.
LDX: »Load X with« lädt das X-Register mit dem angegebenen Wert (unmittelbar) oder mit dem Inhalt der angegebenen Speicherstelle (absolut oder Y-indiziert). Dem Wert entsprechend werden N- und Z-Flag verändert.
LDY: »Load Y with« lädt das Y-Register unmittelbar, absolut oder X-indiziert. Dem Wert entsprechend werden N- und Z-Flag verändert.
LSR: »Logical Shift Right« verschiebt den Inhalt des Akkumulators um ein Bit nach rechts. Das dadurch herausfallende (am weitesten rechts stehende) Bit wandert ins Carry-Flag. Von links wird eine Null in den Akkumulator geschoben. Die Operation entspricht mathematisch einer Division durch Zwei, der anfallende Rest steht im Carry-Flag. Neben dem C-Flag wird eventuell noch das Z-Flag verändert.
NOP: »No operation« ist ein Leerbefehl. Er wird eingesetzt, um Warteschleifen zu bilden oder als Platzhalter für später einzufügende Befehle.
ORA: »Or Accumulator with memory« oder-verknüpft den Akkumulator mit dem angegebenen Wert oder dem Inhalt der angegebenen Speicherzelle. Es können N- und Z-Flag beeinflußt werden.
PHA: »Push Accumulator on Stack« rettet den Akkumulator auf den Prozessor-Stapel. Der Stack-Pointer (Stapelzeiger) wird um einen Zähler vermindert.
PHP: »Push Processorstatus on Stack« rettet das Statusregister und speichert so alle Flags.
PLA: »Pull Accumulator from Stack« holt den Akkumulator-Inhalt vom Prozessor-Stapel, der Stackpointer wird wieder hinaufgezählt, gegebenenfalls werden N- und Z-Flag verändert.
PLP: »Pull Processorstatus from Stack« holt das Statusregister vom Prozessor-Stapel. Es werden alle alten Flags überschrieben!
ROL: »Rotate Left« verschiebt den Inhalt der Speicherzelle beziehungsweise des Akkumulators um ein Bit nach links. Das herausfallende Bit gelangt ins Carry-Flag, der ursprüngliche Carry-Flag-Inhalt wird von rechts hereingeschoben. Der Befehl kann das N-, Z- und C-Flag beeinflussen.
ROR: »Rotate Right« wie ROL, allerdings rechts herum.
RTI: »Return from Interrupt« beendet eine Interrupt- oder BRK-Routine, stellt den alten Wert des Programmzählers und des Statusregisters wieder her.
RTS: »Return from Subroutine« beendet ein Unterprogramm und kehrt zur Verzweigungsstelle zurück, indem der alte Programmzähler vom Stapel geholt wird.
SBC: »Subtract from Accumulator« zieht den angegebenen Wert vom Akkumulator ab. Darüber hinaus wird noch eine 1 abgezogen, wenn das Carry-Flag nicht gesetzt ist. Außerdem wird das Carry-Flag gelöscht, wenn die abgezogene Zahl größer als der Akkumulator-Inhalt war.
SEC: »Set Carry-Flag« wird gesetzt, um bei Subtraktionen richtige Ergebnisse zu erhalten und zu erkennen, ob der abgezogene Wert größer als der Akkumulator-Inhalt war.
SED: »Set Decimal-Flag« setzt das Dezimal-Flag und schaltet damit auf BCD-Arithmetik (BCD bedeutet Binary Coded Decimals). Der maximale Akkumulator-Inhalt beträgt nun 99 anstatt 255, wobei je vier Bit für eine Dezimalzahl zuständig sind.
SEI: »Set Interrupt-Disable-Flag« setzt das I-Flag und verhindert damit weitere rechnergesteuerte Unterbrechungen.
STA: »Store Accumulator in« schreibt den Akkumulator-Inhalt in die angegebene Speicherstelle. Der Akkumulator wird dadurch nicht verändert.
STX: »Store X in« speichert den Inhalt des X-Registers in der angegebenen Speicherstelle.
STY: »Store Y in« speichert den Inhalt des Y-Registers in der angegebenen Speicherstelle.
TAX: »Transfer Accumulator to X« kopiert den Akkumulator-Inhalt in das X-Register. Beeinflußt gegebenenfalls N- und Z-Flag.
TAY: »Transfer Accumulator to Y« kopiert den Akkumulator-Inhalt in das Y-Register. Beeinflußt gegebenenfalls N- und Z-Flag.
TSX: »Transfer Stackpointer to X« überträgt den Stapelzeiger ins X-Register. Kann N- und Z-Flag beeinflussen.
TXA: »Transfer X to Accumulator« Gegenstück zu TAX.
TXS: »Transfer X to Stackpointer« Gegenstück zu TSX.
TYA: »Transfer Y to Accumulator« Gegenstück zu TAY.

Eine komplette Übersicht über alle Befehle mit allen Adressierungsarten und den entsprechenden Codes finden Sie im Tabellenteil dieses Sonderheftes auf Seite 150.

	unmittelbar	absolut	absolut X indiziert	absolut Y indiziert	Zero-Page	Zero-Page indiziert		indiziert indirekt	indirekt indiziert	relativ	impliziert
	#Operator	Op	Op, X	Op, Y	Op	Op, X	Op, Y	(Op, X)	(Op), Y	Op	–
ADC	69	6D	7D	79	65	75	–	61	71	–	–
AND	29	2D	3D	39	25	35	–	21	31	–	–
ASL	–	0E	1E	–	06	16	–	–	–	–	0A
BCC	–	–	–	–	–	–	–	–	–	90	–
BCS	–	–	–	–	–	–	–	–	–	B0	–
BEQ	–	–	–	–	–	–	–	–	–	F0	–
BIT	–	2C	–	–	24	–	–	–	–	–	–
BMI	–	–	–	–	–	–	–	–	–	30	–
BNE	–	–	–	–	–	–	–	–	–	D0	–
BPL	–	–	–	–	–	–	–	–	–	10	–
BRK	–	–	–	–	–	–	–	–	–	–	00
BVC	–	–	–	–	–	–	–	–	–	50	–
BVS	–	–	–	–	–	–	–	–	–	70	–
CLC	–	–	–	–	–	–	–	–	–	–	18
CLD	–	–	–	–	–	–	–	–	–	–	D8
CLI	–	–	–	–	–	–	–	–	–	–	58
CLV	–	–	–	–	–	–	–	–	–	–	B8
CMP	C9	CD	DD	D9	C5	D5	–	C1	D1	–	–
CPX	E0	EC	–	–	E4	–	–	–	–	–	–
CPY	C0	CC	–	–	C4	–	–	–	–	–	–
DEC	–	CE	DE	–	C6	D6	–	–	–	–	–
DEX	–	–	–	–	–	–	–	–	–	–	CA
DEY	–	–	–	–	–	–	–	–	–	–	88
EOR	49	4D	5D	59	45	55	–	41	51	–	–
INC	–	EE	FE	–	E6	F6	–	–	–	–	–
INX	–	–	–	–	–	–	–	–	–	–	E8
INY	–	–	–	–	–	–	–	–	–	–	C8
JMP	–	4C	–	–	–	–	–	–	–	–	–
JSR	–	20	–	–	–	–	–	–	–	–	–
LDA	A9	AD	BD	B9	A5	B5	–	A1	B1	–	–
LDX	A2	AE	–	BE	A6	–	B6	–	–	–	–
LDY	A0	AC	BC	–	A4	B4	–	–	–	–	–
LSR	–	4E	5E	–	46	56	–	–	–	–	4A
NOP	–	–	–	–	–	–	–	–	–	–	EA
ORA	09	0D	1D	19	05	15	–	01	11	–	–
PHA	–	–	–	–	–	–	–	–	–	–	48
PHP	–	–	–	–	–	–	–	–	–	–	08
PLA	–	–	–	–	–	–	–	–	–	–	68
PLP	–	–	–	–	–	–	–	–	–	–	28
ROL	–	2E	3E	–	26	36	–	–	–	–	2A
ROR	–	6E	7E	–	66	76	–	–	–	–	6A
RTI	–	–	–	–	–	–	–	–	–	–	40
RTS	–	–	–	–	–	–	–	–	–	–	60
SBC	E9	ED	FD	F9	E5	F5	–	E1	F1	–	–
SEC	–	–	–	–	–	–	–	–	–	–	38
SED	–	–	–	–	–	–	–	–	–	–	F8
SEI	–	–	–	–	–	–	–	–	–	–	78
STA	–	8D	9D	99	85	95	–	81	91	–	–
STX	–	8E	–	–	86	–	96	–	–	–	–
STY	–	8C	–	–	84	94	–	–	–	–	–
TAX	–	–	–	–	–	–	–	–	–	–	AA
TAY	–	–	–	–	–	–	–	–	–	–	A8
TSX	–	–	–	–	–	–	–	–	–	–	BA
TXA	–	–	–	–	–	–	–	–	–	–	8A
TXS	–	–	–	–	–	–	–	–	–	–	9A
TYA	–	–	–	–	–	–	–	–	–	–	98

Der JMP-Befehl kennt eine weitere Adressierungsart, die indirekte Adressierung. Beim JMP (Op) gibt der Operand ein Zellenpaar an, in dem die tatsächliche Sprungadresse in Low-/Highbyte-Formel steht.

Tabelle 2. Die Assembler-Befehle der 6510-CPU und ihre hexadezimalen Codes in einer Kurzübersicht

Nach dieser Menge von Befehlsdefinitionen werden Sie nun sehen, wie man die Befehle sinnvoll einsetzt. Der Assembler-Anfänger wird zunächst kaum vollständige Maschinencode-Programme schreiben, sondern vielmehr versuchen, kleinere Routinen in Basic-Programme einzubinden. Interessant ist deshalb die Frage, wie man Variablenwerte an die jeweils andere Sprache übergibt. Zu diesem Zweck sind in der Zeropage des C64-Speichers bestimmte Register reserviert, aus denen man die gewünschten Werte direkt auslesen kann. Natürlich ist es ebenfalls erlaubt, beispielsweise den Akkumulatorinhalt durch einen »STA«-Befehl in eine beliebige Speicherstelle zu schreiben und von Basic aus durch »PEEK(Speicherstelle)« zu übernehmen. Einfacher ist es jedoch, den Akkumulatorinhalt, den Inhalt des X-Registers, des Y-Registers und des Statusregisters aus den Adressen 780 bis 783 zu gewinnen. Werte, die von Basic aus übergeben werden sollen, können direkt in diese Speicherstellen gePOKEt werden und befinden sich von diesem Moment an in den Prozessorregistern. Hier hat der Anwender die gewünschten

Werte sofort parat. Umgekehrt ist man in der Lage, über die oben genannten Speicherstellen aus dem Basic-Programm direkt auf den Inhalt des Akkumulators beziehungsweise der anderen Prozessorregister zuzugreifen.

Der Einsatz der Befehle

Die Maschinensprache an sich bietet neben dem Geschwindigkeitszuwachs weitere Vorteile. Der Commodore 64 verfügt über eine Vielzahl verschiedener nützlicher Routinen. Im ROM (Read Only Memory) des Computers sind Maschinensprache-Programme gespeichert, die der C64 zur »täglichen Arbeit« benötigt. Durch die Assembler-Programmierung erhält der Anwender Gelegenheit, diese Routinen für seine Programme zu nutzen.

Es erübrigt sich beispielsweise, Ein- und Ausgabeoperationen zu programmieren, da diese bereits im Kernal des Computers verankert sind und man sie dort nur aufzurufen braucht. Hier gibt es nicht nur die Lade-, Speicher- und Bildschirmoperationen wie von Basic aus, sondern zusätzlich User-Port-Ansteuerungen und direkte Zugriffsoptionen auf die Datenübertragungskanäle. Um diese Funktionen auszuschöpfen, ist es allerdings unerläßlich, sich eines ausführlichen ROM-Listings zu bedienen. Dadurch erhält man einerseits einen guten Zugang zur Funktionsweise des Computers, andererseits sind alle wichtigen Routinen für eigene Programme umfassend inklusive ihrer Adressen aufgeführt. Sie bieten außerdem die Gelegenheit, viel über die professionelle Assemblerprogrammierung zu lernen. Mehr über diese Programmiertechnik erfahren Sie in dem Kurs »Von Basic zu Assembler« in diesem Sonderheft. Eine weitere Spezialität der Maschinensprache ist das Ausnutzen des Interrupts.

Unser Computer ist – solange er eingeschaltet ist – ständig mit irgendwelchen Tätigkeiten beschäftigt. Im Direktmodus hängt er beispielsweise meistens in einer Warteschleife und harrt der Eingaben, im Programm-Modus arbeitet er sich mit Hilfe der Interpreterschleife durch einen Basic-Befehlstext hindurch und so weiter. Nun werden Sie ja sicher schon festgestellt haben, daß er im Direktmodus auch den Cursor blinken läßt, in beiden Modi die TI$-Uhr weiterzählt und weitere Dinge macht, die anscheinend so nebenher passieren. Der Mensch kann mehrere Dinge gleichzeitig tun, der Mikroprozessor ist nur fähig zu einer Arbeit pro Zeiteinheit. Weil aber diese Zeiteinheiten so unfaßbar kurz sind (etwa eine Millionstel Sekunde), haben wir Benutzer den Eindruck der Gleichzeitigkeit.

Kurze Unterbrechung

Wenn dem aber so ist, wie macht es der Computer, daß er beispielsweise ein Programm abarbeitet und trotzdem die TI$-Uhr weiterzählt? Durch Unterbrechungen (interrupt = unterbrechen) der gerade ausgeübten Tätigkeit. Ein Beispiel aus dem täglichen Leben soll uns das illustrieren: Sie lesen gerade diesen Artikel, da klingelt das Telefon und ein Freund möchte von Ihnen wissen, was eigentlich Interrupts sind. Während Sie es ihm erklären, fängt in der Küche der Teekessel schrill zu pfeifen an. Sie sagen Ihrem Freund, er möge sich einen Moment gedulden, gehen in die Küche und nehmen den Kessel vom Feuer. Dann kehren Sie ans Telefon zurück und beenden nach einer Weile das Gespräch. Nach dem Auflegen des Telefonhörers setzen Sie die Lektüre des Artikels fort, fest entschlossen, sich nun nicht mehr unterbrechen zu lassen. Kurze Zeit später klingelt jemand an der Tür. Sie lassen sich dadurch aber nicht mehr stören.

Dieses Gleichnis gibt ziemlich genau wider, was sich im Computer – nur bei millionenfacher Geschwindigkeit – bei Unterbrechungen abspielt. In Bild 4 ist das Schema des Ablaufs grafisch dargestellt. In gewisser Weise ähnelt das ganze dem Abarbeiten von Unterprogrammsequenzen. Weshalb programmiert man dann nicht einfach mittels einiger JSR-Aufrufe? Dafür hat L.A.Leventhal einen einleuchtenden Vergleich: »Ein Unterbrechungs-System entspricht etwa einer Telefonklingel. Sie läutet, wenn ein Anruf empfangen wird, so daß man den Hörer nicht laufend abnehmen muß, um festzustellen, ob sich jemand in der Leitung befindet.« (L.A.Leventhal, »6502-Programmieren in Assembler«, te-wi Verlag München, Seite 12-1). Unterbrechungen können dann angefordert und abgearbeitet werden, wenn sie nötig sind, im Gegensatz zu Unterprogrammen, die erst dann berücksichtigt werden, wenn der Programmzähler einen JSR-Befehl erfaßt. Um also schnell reagieren zu können, müßte man sehr oft in einem Programm eine Unterroutine anspringen, die gewisse Registerinhalte prüft und dann zur Bearbeitung verzweigt oder – bei Nichtvorliegen einer Bedingung – im normalen Programm weiterfährt. Das kostet unnötig Zeit und Speicherraum. Mancher Verkehr des Computers mit Peripherie erfordert so schnelle Reaktionen, daß diese nur durch Unterbrechen des laufenden Programmes geleistet werden können.

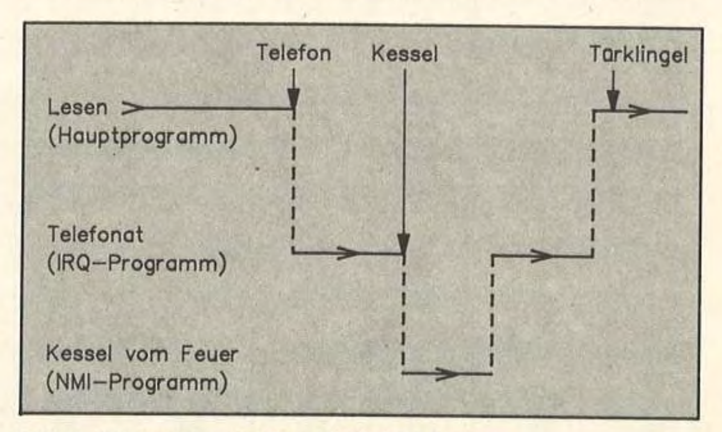


Bild 4. Interrupts aus dem täglichen Leben

Ich denke, daß Sie nun die Notwendigkeit von Unterbrechungen erkennen. Fast jede CPU kennt solche Unterbrechungssysteme. Der Interrupt (Unterbrechung) ist eine vom Betriebssystem in einem bestimmten Rhythmus automatisch ausgeführte Operation, die das gerade laufende Programm unterbricht und bestimmte Register abfragt, ob irgendwelche Ereignisse eingetreten sind, die ein programmunabhängiges Eingreifen des Computers erfordern. Das beste Beispiel ist die Benutzung der RUN/STOP-RESTORE-Tastenkombination. Drückt man während eines Programmablaufs eine Taste, so bleibt das in der Regel ohne Einfluß auf das Programm, der Computer läßt sich nicht stören, sondern fährt mit der Programmbearbeitung fort. Anders liegt der Fall jedoch, wenn man die RUN/STOP-RESTORE-Kombination benutzt. Der Computer unterbricht das Programm und meldet sich »zurück«. Woran liegt das? Die Lösung ist, daß während des Interrupts unter anderem geprüft wird, ob diese Tasten benutzt werden. Sobald der Computer erkennt, daß der Anwender das Programm unterbrechen will, verzweigt die Interrupt-Routine und führt einen Warmstart aus.

Dieser »Systeminterrupt« erfolgt automatisch und ist von Basic aus nicht zu beeinflussen. Man kann zwar durch einen geeigneten POKE die RESTORE-Taste abschalten und so einen Programmabbruch verhindern, der Interrupt selbst erfolgt jedoch weiterhin, wenn auch unsichtbar.

Wenn nun jemand den Wunsch verspürt, diese automati-

sche Programmunterbrechung für seine Zwecke zu nutzen, so ist das durchaus verständlich. Interruptprogrammierung eröffnet eine große Zahl interessanter Anwendungen. Joystick- und Tastaturabfragen lassen sich damit effektiver realisieren. Allerdings ist die Nutzung des Interrupts von Basic aus nahezu unmöglich.

In weiser Voraussicht haben die Programmierer der C64-Systemroutinen einige Interruptregister bereitgestellt, die von Basic aus durch POKEs gesteuert werden können. Zum Beispiel das Sprite-Kollisions-Register, das – unabhängig vom Programmablauf – je nach Wunsch eine Sprite-Sprite- oder Sprite-Hintergrund-Kollision erkennt und ein entsprechendes Bit setzt, das dann vom Programm aus weiter behandelt werden kann.

Der Commodore 64 verfügt über vier verschiedene Interruptarten:

1. Der Reset.

Dieser Interrupt wird einerseits direkt nach dem Einschalten des Computers ausgelöst und ist verantwortlich für die Einschaltmeldung. Andererseits kann man diesen Interrupt auch erzeugen, indem man ihn über die Reset-Leitungen am seriellen oder parallelen Bus anfordert, ein Prinzip, das von den Resettastern angewandt wird. Der Reset kann nicht verhindert werden. Er wird in jedem Fall ausgeführt, wenn auch manchmal ohne Ergebnis. Bei Auslösung des Resets wird nämlich zunächst die Speicherstelle $8000 (Modulstart mit CBM80-Kennung) abgefragt.

Verschiedene Interrupt-Arten

Steht dort die Anfangsadresse eines Programms, so wird die Resetroutine unterbrochen und das Programm neu gestartet.

2. Der nicht maskierbare Interrupt (NMI).

Hier handelt es sich um den oben erwähnten RESTORE-Tasten-Interrupt. Außerdem ist er für die Ansteuerung der RS232-Schnittstelle zuständig. Bei Auslösung dieses Interrupts erfolgt ein Sprung zu der Adresse, die in den Speicherstellen 792 und 793 steht.

3. Der BRK-Interrupt (Break).

Wird durch den Assembler-Befehl »BRK« ausgelöst und springt daraufhin zur Adresse, die im Vektor in den Speicherstellen 790 und 791 angegeben ist (normalerweise $FE66).

4. Der maskierbare Interrupt (IRQ).

Dieser Interrupt ist softwaremäßig steuerbar. Er kann also vom Programm aus an- und abgeschaltet werden. Die Assemblerbefehle »SEI« und »CLI« sind dazu vorgesehen. Dieser Interrupt wird in der Regel von den Timern der CIA gesteuert und nach jeder 1/60 Sekunde ausgelöst. Es wird in die IRQ-Routine verzweigt (Vektor in Adresse in 788 und 789). Dort wird die Tastatur abgefragt (beispielsweise, ob die RUN/STOP-Taste bedient wurde), das Cursorblinken veranlaßt und die interne Uhr weitergezählt. Außerdem gibt es noch ein Register, das diesen Interrupt beeinflussen kann. Es handelt sich hierbei um das Interrupt Request-Register 53273 im VIC. Das Register bietet insgesamt vier verschiedene Optionen für die Auslösung eines Interrupts an:

Rasterzeileninterrupt
Lightpeninterrupt
Sprite-Sprite-Kollision
Sprite-Hintergrund-Kollision

Welche Vorteile bietet nun die Assembler-Programmierung zur Ausnutzung der Interruptroutinen? Die Adresse in den Speicherstellen 788 und 789 ist veränderbar, das heißt man kann anstelle der vorhandenen IRQ-Prozedur (Tastaturabfrage, interne Uhr, Cursorblinken) ein eigenes Maschinenprogramm in den Interrupt einbinden.

Programme laufen parallel

Man muß nur den sogenannten IRQ-Vektor (die Adresse in 788/789) verbiegen, also die Startadresse des eigenen Programms in diese Speicherstellen schreiben. Allerdings fragt der Computer alle 1/60 Sekunde diese Speicherstellen ab, und wenn man gerade daran herummanipuliert, führt das zu einem Absturz des Systems. Man muß also zuvor den Interrupt ausschalten. Von Assembler aus ist das durch Setzen des »Interrupt Disable Flags« vollziehbar:

```
SEI
```

Danach richtet man den IRQ-Vektor auf die Startadresse des eigenen Programms (beispielsweise $C000):

```
LDA #  $00 ; Low-Byte
STA $0314 ; dezimal 788
LDA#$C0 ; High-Byte
STA $0315 ; dezimal 789
```

Ab jetzt ist der Interrupt wieder zulässig, wir geben ihn also frei:

```
CLI
```

Allerdings ist darauf zu achten, daß bei $C000 (dezimal 49152) tatsächlich ein lauffähiges Programm steht. An der angegebenen Adresse kann zum Beispiel eine Joystickabfrage beginnen, die nun interruptgesteuert ständig abgefragt wird. Steuerbewegungen werden sofort und unverzüglich erkannt und übertragen, ohne daß die Abfrage innerhalb des ursprünglichen Programms aufgerufen werden muß.

Hier noch einige grundsätzliche Bemerkungen zu Maschinenspracheprogrammen:

Assemblierter Quellcode wird als reiner Maschinencode auf Datenträger gespeichert. Wenn man diesen wieder lädt, darf das nicht durch »LOAD"Programmname"(,1)« beziehungsweise »LOAD"Programmname",8« geschehen. In diesem Fall würde das Programm an den Anfang des Basic-Speichers ($0800) geladen. Dort steht nun ein Maschinenprogramm mit Sprungadressen, die sich auf einen ganz anderen Bereich beziehen. Man muß also Sorge tragen, daß das Programm direkt in den gewünschten Speicherbereich geladen wird. Das geschieht durch Anfügen der Sekundäradresse 1 an den Ladebefehl, also »LOAD"Programmname",1,1« beziehungsweise »LOAD "Programmname",8,1«. Das Programm wird danach durch »SYS"Anfangsadresse"« gestartet. Zuvor sollte man ein »NEW« eingeben, damit verstellte Basic-Vektoren wieder gerichtet werden. Andernfalls erhält man bei Programmierversuchen in Basic einen OUT OF MEMORY ERROR.

Starten von Maschinenprogrammen

Maschinenspracheprogramme – soweit sie nicht im Kassettenpuffer stehen – werden durch einen Reset nicht gelöscht, sie können durch den entsprechenden »SYS«-Befehl neu gestartet werden. Außerdem gehen in der Regel keine Assembler-Programme durch das Nachladen eines Basic- oder Maschinenspracheprogramms verloren, es sei denn, der Speicherbereich, in dem sie sich befinden, wird von dem neuen Programm überschrieben. Um dies zu vermeiden, werden Maschinenspracheprogramme oft in den Speicherbereich von $C000 bis $CFFF gelegt, da dieser Teil des C64-Speichers einerseits dem Benutzer uneingeschränkt zur Verfügung steht, andererseits aber vor dem Überschreiben durch Basic-Programme geschützt ist.

Die Register des 6510-Prozessors stellen uns zum Rechnen nur die 256 verschiedenen Zahlen, die durch acht Bit darstellbar sind, zur Verfügung. Man kann entweder die Zahlen von 0 bis 255 oder von -128 bis +127 benutzen (vorzeichenbehaftete Arithmetik. Ist das achte Bit gesetzt, so bedeutet dies eine negative Zahl). Natürlich reichen diese Zahlen für vernünftige Rechenoperationen nicht aus.

Die Behandlung von Zahlen

Allein bei der Rechnung mit ganzen Zahlen werden oft drei- und vierstellige Werte benutzt. Das Entscheidende dabei ist jedoch, daß der Computer auch Zahlen kleiner als eins verarbeiten kann. Wie aber kann man solche Berechnungen ausführen mit einem Register, das maximal 256 verschiedene Zahlen zur Rechnung bereitstellt? Die Antwort auf diese Frage führt in das Gebiet der 16-Bit- und Fließkomma-Arithmetik.

Die Darstellung von ganzen Zahlen größer als 255 und kleiner als -128 bewerkstelligt der Prozessor auf eigene Art: Die Zahl wird einfach in zwei Byte aufgespalten. Dabei ist die Verfahrensweise so, daß sobald das erste Byte die Grenzen seiner Aufnahmefähigkeit erreicht hat, das zweite Byte um eins hochgezählt und das erste Byte gelöscht wird. Ein Assemblerprogramm zur Addition der Zahlen 250 und 100 könnte also wie folgt aussehen:

```
        LDX # 250
        LDY # 100
START   INX
        CPX # 0
        BNE MARKE
        INC Highbyte-Register
MARKE   DEY
        CPY # 0
        BNE START
        STX Lowbyte-Register
```

Die zu addierenden Zahlen werden in den Zählregistern des Prozessors abgelegt. Eine Zahl wird vermindert, die andere entsprechend erhöht. Zwischendurch wird geprüft, ob eines der Zählregister den Wert Null erreicht hat. Sollte das vermindernde Register (hier Y) den Wert Null erreichen, so ist die Addition beendet.

Wenn zuvor das X-Register den Wert 255 überschreitet, entsteht ein Überlauf, der im »Highbyte-Register« abgelegt wird. Das X-Register wird daraufhin automatisch zurückgesetzt und der Prozeß läuft weiter, bis der Inhalt des Y-Registers Null ist.

16-Bit-Arithmetik

Die oben behandelte Vorgehensweise ist zwar universell einsetzbar, jedoch gibt es wesentlich geschicktere Lösungen, um eine Addition und Subtraktion vorzunehmen. Der Prozessor versteht nämlich die Befehle ADC (Add with Carry) und SBC (Subtract with Carry). Bei Verwendung von ADC werden sofort Akkumulatorinhalt und angegebene Speicherstelle addiert, und sobald ein Überlauf auftritt, wird das Carry-Flag gesetzt. Arbeitet man mit vorzeichenbehafteter Arithmetik, so wird das Overflow-Flag gesetzt, wenn der zulässige Bereich überschritten wird. Diese Flags sollten also vor einer Additions-Operation gelöscht werden. Bei der Subtraktion hingegen wird an einem gelöschten Carry-Flag erkannt, ob ein Unterlauf stattgefunden hat. Dieses Flag sollte also vor einer Subtraktion gesetzt werden. Falls das Carry-Flag nicht gesetzt war, wird noch eine 1 vom Ergebnis abgezogen.

Das Ergebnis liegt jetzt also in einer Low-/High-Byte-Aufschlüsselung vor. Die Werte der einzelnen Bits des High-Bytes sind nun:

Bit 15	Bit 14	Bit 13	Bit 12
32768	16384	8192	4096
Bit 11	Bit 10	Bit 9	Bit 8
2048	1024	512	256

Die 16-Bit-Darstellung der Zahl 250+100=350 lautet:
0000000101011110 = 256+64+16+8+4+2

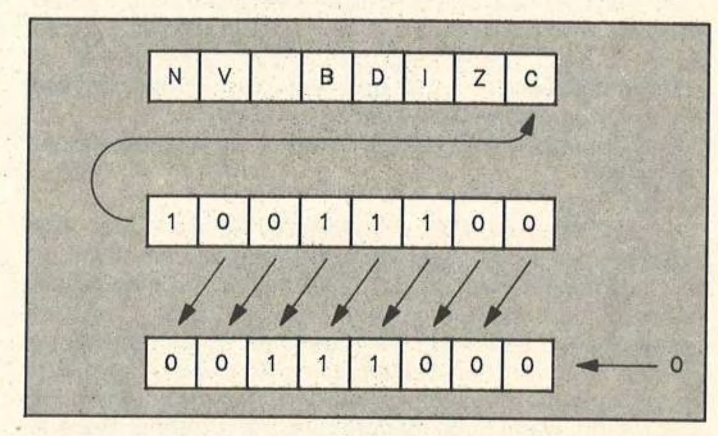


Bild 5. Der ASL-Befehl: Die Bits einer Speicherstelle werden nach links geschoben. Bit 7 landet im Carry-Flag.

Bei der Verwendung von 16 Bit können insgesamt 65536 verschiedene Zahlen dargestellt werden. Rechnet man mit vorzeichenbehafteter Arithmetik, so erhält Bit 15 den Wert -32768. Der Zahlenbereich erstreckt sich nun von -32768 bis +32767. Diese Zahl wird vielen bekannt vorkommen. Tatsächlich ist dies nämlich genau der Bereich, den der C64 bei Rechnungen mit ganzen Zahlen (Integers) abdecken kann.

Fließendes Komma

Interessant ist das Verfahren zur Darstellung von Zahlen, die kleiner als eins sind. Die verschiedenen Bits werden ähnlich eingesetzt wie bei der »normalen« Darstellung:

Normal:

Bit 0 = 2^0 = 1
Bit 1 = 2^1 = 2
Bit 2 = 2^2 = 4
Bit 3 = 2^3 = 8
Bit 4 = 2^4 = 16
Bit 5 = 2^5 = 32
Bit 6 = 2^6 = 64
Bit 7 = 2^7 = 128
Bit 8 = 2^8 = ...

Kleiner als eins:

Bit 31 = 2^{-1} = $1/2^1$ = 1/2
Bit 30 = 2^{-2} = $1/2^2$ = 1/4
Bit 29 = 2^{-3} = $1/2^3$ = 1/8
Bit 28 = 2^{-4} = $1/2^4$ = 1/16
Bit 27 = 2^{-5} = $1/2^5$ = 1/32
Bit 26 = 2^{-6} = $1/2^6$ = 1/64
Bit 25 = 2^{-7} = $1/2^7$ = 1/128
.
.
.
Bit 0 = 2^{-32} = $1/2^{32}$

Auffällig ist, daß für die Darstellung von natürlichen Zahlen vier Byte im C64 zur Verfügung stehen. Durch diese Vorgehensweise kann man natürlich ungleich größere Genauigkeiten erreichen. Ergänzt man die vier Bytes nun

um ein weiteres, das als Exponent dient, so lassen sich mit dem C64 Zahlen von etwa -1E38 bis +1E38 bilden. 1E38 bedeutet 1×10^{38}, das ist eine Eins mit 38 Nullen. Diese Zahl ist wesentlich größer als die Zahl 32767, die beim Rechnen im Bereich der ganzen Zahlen zur Verfügung steht. Natürlich ist die Speicherung der Fünf-Byte-Zahl aufwendiger. Deshalb bietet der Basic-Interpreter auch an, eine Variable als Integer- oder Fließkommazahl (gleich reelle Zahl) zu definieren und dem Prozessor so beim Ab-

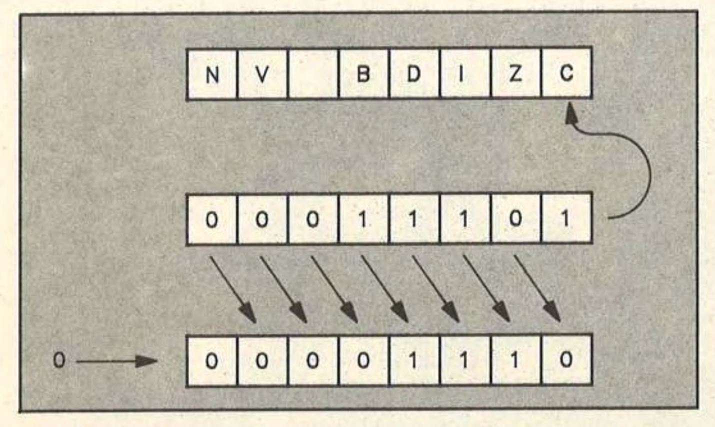


Bild 6. Der LSR-Befehl: Verschiebung der Bits einer Speicherstelle nach rechts. Bit 0 landet im Carry-Flag.

arbeiten des Wertes unnütze Arbeit zu ersparen. Hierzu erfahren Sie im Kurs »Von Basic zu Assembler« mehr.

Die Maschinensprache stellt Befehle zur Verfügung, die Sie in die Lage versetzen, einfache Multiplikations- und Divisionsaufgaben zu lösen. Durch ein durchgehendes Verschieben der Bitwerte im Prozessorregister kann man beispielsweise eine Multiplikation mit zwei erreichen (Bild 5):

00011100 = 28 ASL (Arithmetical Shift Left): 00111000 = 56 = 2*28

Die ASL-Operation verschiebt also die Bits nach links und füllt Bit Null mit einer Null auf. Das Bit, das links herausfällt, gelangt ins Carry-Flag. Sollte also der Fall auftreten, daß das siebte Bit gesetzt ist, so kann man es aus dem Carry-Flag ins »Highbyte-Register« übertragen:

10011100 = 156
ASL: 00111000 = 56
+
00000001 im »Highbyte-Register« = 256
56+256 = 312 = 2*156

Bei Ausführung dieser Operation sollte man darauf achten, das Carry-Flag vorher zu löschen, damit das Ergebnis nicht durch ein zufällig gesetztes Flag verfälscht wird.

Wenn man nun Multiplikationen mit anderen Zahlen, beispielsweise 10 (10 x Inhalt) ausführen will, kann man so vorgehen, daß man zunächst das Register dreimal mit zwei multipliziert (2x2x2x Inhalt = 8x Inhalt) und dazu noch zweimal den ursprünglichen Registerinhalt hinzuaddiert. Die Vorgehensweise ist natürlich aufwendig, aber schneller, als den Registerinhalt zehnmal aufzuaddieren. Genauso wertvoll wie der ASL-Befehl ist der LSR-(Logical Shift Right-) Befehl. Er erlaubt eine Division durch zwei (Bild 6):

00011100 = 28
LSR: 00001110 = 14

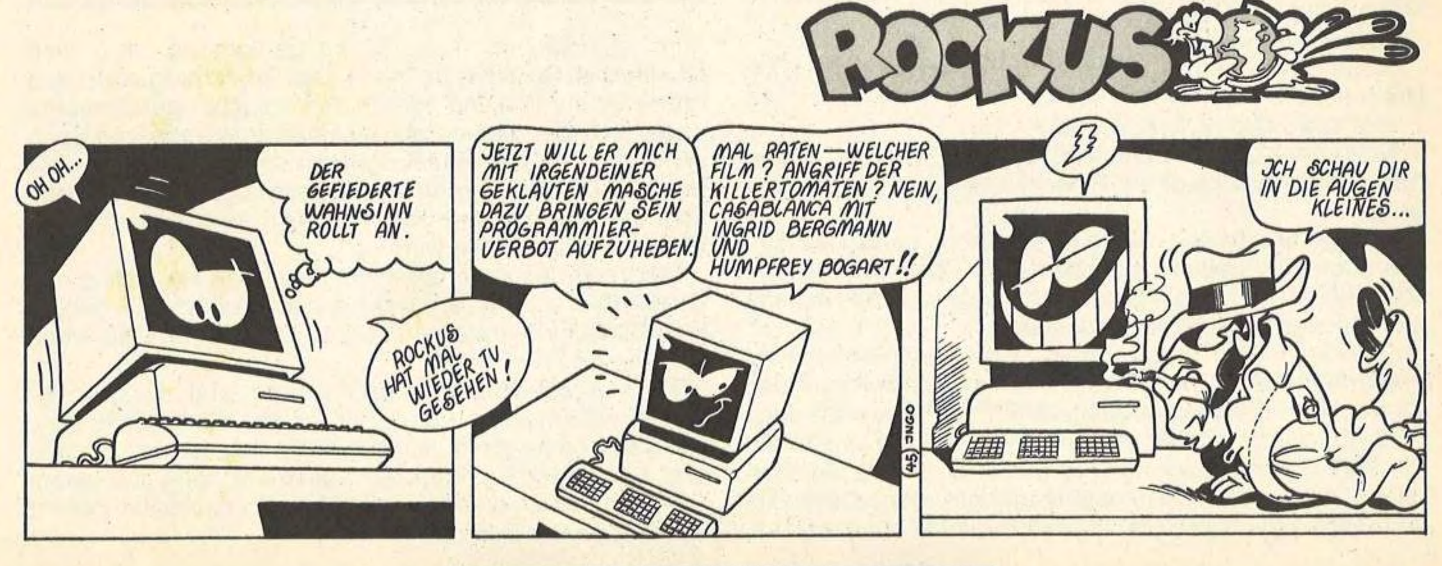
Hier wird Bit Null ins Carry-Flag gerettet und das siebte Bit mit einer Null aufgefüllt:

00011101 = 29
LSR: 00001110 = 14

An dieser Stelle tritt ein Problem auf. Beim Rechnen mit ganzen Zahlen existiert keine Nachkommastelle. Das heißt, eine Division durch zwei ergibt nur den ganzzahligen Ergebniswert, eine eventuell vorhandene Nachkommastelle kann nicht ohne weiteres im Programmablauf weiterverwendet werden. Allerdings ist das Überprüfen des Carry-Flags gut geeignet, um die Zahl auf gerade oder ungerade zu prüfen. Bei der Division wird man also Rundungsfehler in Kauf nehmen oder auf die Fließkomma-Arithmetik ausweichen. Zahlen unter Fließkomma-Arithmetik besitzen beim C64 einen speziellen Arbeitsbereich, den FAC (Floating Point Accumulator). In der Zeropage des C64 sind bestimmte Speicherstellen für die Berechnung einer Fließkomma-Zahl freigehalten. In Register 97 steht der Exponent der Zahl, in den Registern 98 bis 101 befindet sich die Mantisse und in Register 102 befindet sich der Vorzeichenwert. In den Speicherstellen 105 bis 110 befindet sich ein weiterer Fließkomma-Akkumulator mit identischer Belegung. Die Zahl 29 im Fließkommaformat hätte die Gestalt 0,29E02, das heißt, 0,29x100 = 29. Man muß also die Zahl in Fließkomma-Bits aufspalten. Zunächst ziehen wir die größtmögliche Fließkomma-Zahl vom Ausgangswert ab, in diesem Fall 0,25. Bit 31 muß also gesetzt sein. Übrig bleibt der Wert 0,04 =1/25. Der nächst kleinere Fließkomma-Wert ist 1/32 und so weiter.

Auch diese Thematik wird noch ausführlich im Kurs »Von Basic zu Assembler« in diesem Sonderheft behandelt. Der Einstieg sollte Ihnen jedoch mit den vorhandenen Informationen gelingen.

(Uli Eicke/Dr. Rudolf Egg)



Ein neuer Standard: »Giga-Ass«

Hypra-Ass bekommt einen würdigen Nachfolger: »Giga-Ass«. Durch optimale Speicherausnutzung wird dieser komfortable Makro-Assembler zum idealen Werkzeug für Maschinen-Programmierer. Volle Kompatibilität zu Hypra-Ass Quellcodes ist durch ein Konvertierungsprogramm garantiert.

Hervorgegangen ist Giga-Ass aus dem Programm »Hypra-Ass«, veröffentlicht in der Ausgabe 7/85 des 64'er-Magazins und im Sonderheft 8/85. Dieses Programm wurde nun in den vergangenen zwei Jahren perfektioniert. Das Ergebnis ist Giga-Ass.

Der Assembler Giga-Ass belegt den ROM-Bereich von $8000 bis $9FFF und wird wahlweise im RAM betrieben oder über ein EPROM-Modul eingeblendet. Listing 1 stellt also eine brennfähige Version mit Modulsimulation dar. In Tabelle 1 finden Sie eine Übersicht mit den wichtigsten Merkmalen von Giga-Ass.

Starten des Assemblers

a) Sie verwenden das Original-Betriebssystem des C64. Dann geben Sie ein:

```
LOAD "GIGA-ASS",8,1 <RETURN>
```

Nach dem Laden SYS 64738 und <RETURN> eingeben. Darauf erscheint die Einschaltmeldung des Assemblers.

Verfügt Ihr C64 über einen Reset-Knopf, genügt es, diesen nach dem Laden zu drücken. Da Giga-Ass durch die »CBM80«-Kennung ein Modul simuliert, wird der Assembler bei einem Reset sofort gestartet.

Giga-Ass kann auch direkt auf ein 2764-EPROM gebrannt werden. Die EPROM-Platine muß dann auf den Speicherbereich von $8000 bis $9FFF adressiert werden.

Als Quelltextspeicher stehen 30 KByte RAM von $0800 bis $7FFF zur Verfügung. Als Speicher für die Symboltabelle (Labels, Makronamen) dient der unter dem Basic-Interpreter befindliche RAM-Bereich von $A000 bis $BFFF. Dieser nimmt 1170 Symbole auf, da aufgrund des besonders kompakten Formats jedes Symbol nur 7 Byte belegt. Der Speicher $C000 bis $CFFF bleibt für einen Monitor frei. Giga-Ass und der SMON können sich also gleichzeitig im Speicher befinden.

Der Quelltext wird wie ein Basic-Programm mit dem zeilenorientierten Editor von Basic eingegeben und in einem kompakten Format im Speicher abgelegt. Sämtliche Assemblerbefehle werden in Form von »Token« kodiert, das heißt, jeder Assembler-Befehl braucht nur 1 Byte Speicherplatz. Außerdem lassen sich alle Befehle auch abgekürzt eingeben; in den meisten Fällen genügt dafür eine einzige Taste. Nach Eingabe einer Zeile wird diese sofort auf dem Bildschirm in Klartext und formatiert angezeigt.

Komfortabler Editor

Zum eigentlichen Assemblierungs-Vorgang muß sich Quelltext im Speicher befinden. Das Objektprogramm wird entweder auf Diskette oder in den Speicher geschrieben, wobei sich durch eine zusätzliche Code-Adresse der Speicherbereich, in dem das Programm abgelegt wird, frei wählen läßt. Die Erzeugung des Objektprogramms läßt sich aber auch ganz unterdrücken, damit nur ein reiner Syntax-Check vorgenommen wird.

Während des Assemblierens kann man ein Listing des Quelltextes auf den Drucker oder auf Diskette ausgeben. Eine Bildschirmausgabe ist bei 40 Zeichen pro Zeile weniger sinnvoll.

Nach Abschluß der Assemblierung wird die Symboltabelle wahlweise auf dem Bildschirm, auf Diskette oder Drucker ausgegeben. Alle Symbole werden mit Namen und Symbolwert aufgelistet. Außerdem steht bei jedem Symbol die Zeilennummer der Zeile, in der es verwendet wurde, und der Symboltyp.

GIGA-ASS unterscheidet drei Typen von Symbolen:
1. Makronamen
2. Globale Label
3. Lokale Label

Makros können an jeder Stelle des Quelltextes definiert und verwendet werden. Ein Makroaufruf unterscheidet sich in nichts von einem Assemblerbefehl: Es wird nur der Makroname eingegeben. Eventuelle Parameter werden von dem Makro-Namen durch ein Leerzeichen (<SPACE>) und untereinander durch Kommata getrennt.

Die Eingabe von Qelltextzeilen

Eine Quelltextzeile wird wie eine Basic-Zeile eingegeben. Damit der Editor die einzelnen Elemente einer Quelltextzeile (Zeilennummer, Label, Assembler-Befehl und Kommentar) voneinander unterscheidet, müssen Sie einige Regeln beachten:

1. Jede Quelltextzeile beginnt mit einer eindeutigen Zeilennummer. Diese wird entweder direkt eingegeben oder mit dem Auto-Befehl vorgegeben.

2. Unmittelbar nach den Zeilennummern folgen die Label. Zwischen Zeilennummer und Label darf kein Leerzeichen stehen.

3. Ein auf ein Label folgender Assemblerbefehl muß vom Label durch ein Leerzeichen getrennt werden.

4. Enthält eine Zeile kein Label, so beginnt sie mit einem Leerzeichen nach der Zeilennummer, welches unbedingt eingegeben werden muß.

Programmieren – einfach wie in Basic

5. Kommentare werden am Ende der Zeile angefügt und vom Rest der Zeile durch ein Semikolon getrennt. Reine Kommentarzeilen haben als erstes Zeichen in der Zeile ein Semikolon.

6. Enthält eine Quelltextzeile nur ein Label und keine Assembleranweisung, so darf auf das Label kein Kommentar folgen!

Die Editor-Befehle

Da der Assembler-Editor eine Erweiterung des Basic-Editors darstellt, sind die normalen Basic-Befehle wie OPEN, CLOSE, CMD, PRINT#, POKE, PEEK und SYS wie gewohnt verwendbar. Nicht erlaubt sind alle Basic-Befehle, die Variablen anlegen, also LET, FOR, NEXT, DIM etc.

Die neu hinzugekommenen Editor-Befehle sind im Gegensatz zu den Basic-Befehlen nur ein Zeichen lang. Darauf ist bei Überschneidungen zu achten. Zum Beispiel heißt der Editor-Befehl zum Laden von Quelltexten »L«, während der Basic-Befehl bekanntlich LOAD lautet.

Die Unterscheidung zwischen Basic- und den neuen Befehlen wird vorgenommen, indem geprüft wird, ob das zweite Zeichen einer Eingabe ein (eventuell geSHIFTeter) Buchstabe ist. Dies ist bei Basic-Befehlen immer der Fall, bei den anderen Befehlen ist das zweite Zeichen eine Zahl oder ein Hochkomma.

Steckbrief Giga-Ass
- 30 KByte Quelltextspeicher
- Quelltexte kompatibel zu Hypra-Ass (mit Konvertierprogramm)
- Komfortable Funktionstastenbelegung
- Kompakter Quellcode durch Token-Verarbeitung
- Programm liegt in brennfähiger Version vor (für EPROM-Module)
- Modulstart integriert (CBM80)
- Resetfest
- Platz für Monitor ab $C000
- OLD-Funktion
- Assemblieren direkt auf Diskette
- Formatierte Ausgabe von Symboltabellen
- Merge-Funktion für Quelltexte
- Basic-Befehle im Direktmodus weiterhin nutzbar
- Bis zu 1170 Symbole (Label, Makronamen, etc.) verwendbar

Tabelle 1. »Giga-Ass« auf einen Blick

21 Editor-Befehle

A (Auto)

dient zur automatischen Vorgabe der Zeilennummern bei der Eingabe neuer Quelltextzeilen. Syntax:

A Zeilennummer, Schrittweite

Zum Beispiel gibt A 100,10 die Zeilennummern ab 100 in Zehnerschritten vor (100, 110, 120, 130 etc.). Die Numerierung wird abgebrochen, indem man mit <SHIFT RETURN> den Cursor in eine leere Zeile bewegt und <RETURN> drückt. Gibt man im Auto-Modus direkt hinter einer Zeilennummer <RETURN> ein, so wird diese übersprungen und sofort die nächste Zeilennummer ausgegeben. Durch Drücken von <RETURN> können so größere Bereiche übersprungen werden. Befinden Sie sich im Auto-Modus, so lassen sich bereits eingegebene Zeilen ändern, ohne den Auto-Modus zu verlassen. Voraussetzung ist jedoch, daß die betreffenden Zeilen sich noch auf dem Bildschirm befinden. Man setzt die Numerierung fort, indem man in der Quelltextzeile mit derjenigen Zeilennummer, die zuletzt vorgegeben wurde, <RETURN> drückt.

Die Numerierung kann nach Abbruch automatisch an der alten Stelle wieder aufgenommen werden durch Drücken von <A> und <RETURN>.

B (Speicheranzeige) <F2>

zeigt den gesamten für den Quelltext zur Verfügung stehenden und den davon bereits benutzten Speicherbereich in KByte an. Zusätzlich wird die Anzahl der belegten und noch freien Bytes ausgegeben. Der B-Befehl wird auch durch <F2> aufgerufen.

C (Kaltstart)

bewirkt einen Neustart des Assemblers. Die Speicherkonfiguration wird nicht verändert. Es handelt sich nicht um einen Reset, sondern um eine Re-Initialisierung. Mit <O> (Old) und <RETURN> wird der Quelltext zurückgeholt.

D (Delete)

dient zum Löschen von Quelltextzeilen. Syntax:

D Zeilennummer 1 - Zeilennummer 2

Lassen Sie Zeilennummer 1 oder Zeilennummer 2 fortfallen, so wird von Quelltextanfang beziehungsweise bis Quelltextende gelöscht. Zum Beispiel bewirkt D -100, daß alle Zeilen bis einschließlich Zeile 100 gelöscht werden, und D 1000 -, daß alle Zeilen ab Zeile 100 einschließlich gelöscht werden.

D Zeilennummer löscht eine einzelne Zeile. Alternativ dazu gibt man die Zeilennummer allein und danach <RETURN> ein.

»D« ohne Argument löscht den gesamten Quelltext und entspricht damit dem weiterhin verwendbaren Basic-Befehl NEW.

Vor Ausührung des Löschbefehls erfolgt mit »ARE YOU SURE?« stets eine Sicherheitsabfrage. Nach Bestätigung mit <Y> wird die Operation ausgeführt; jede andere Taste führt zum Abbruch des Befehls.

E (List) <F1>

dient zum Ausgeben von Quelltext. Syntax:

E Zeilennummer 1 - Zeilennummer 2

Wie beim D-Befehl können Zeilenbereiche, einzelne Zeilen oder der gesamte Quelltext geLISTet werden. Der E-Befehl ohne Argumente wird auch durch <F1> aufgeru-

fen. Statt des E-Befehls ist auch das Basic-Befehlswort LIST verwendbar.

Mit <SPACE> wird das LISTen angehalten. Durch erneutes Drücken von <SPACE> fährt das LISTen fort. Mit <RUN/STOP> wird der Vorgang abgebrochen.

Eine Ausgabe des Quelltextes auf einen Drucker erreicht man durch:

```
OPEN1,4,7 : CMD1 <RETURN>
E <RETURN>
```

F (Find)
ermöglicht das Auffinden von Zeichenfolgen im Quelltext. Syntax:

F P,"Zeichenfolge"

Ein Beispiel: F0,"LDA" findet jedes Auftreten des Assembler-Befehls LDA im Quelltext und listet die entsprechenden Zeilen auf. Diese Auflistung wird durch Drücken der SPACE-Taste angehalten und danach durch nochmaliges Drücken von <SPACE> fortgesetzt. Die Auflistung brechen Sie dadurch ab, daß Sie sie mit <SPACE> anhalten und danach <RUN/STOP> drücken.

Der Parameter »P« (0 bis 15) dient zur Angabe der Page, also eines Bereiches im Arbeitsspeicher (siehe hierzu auch die entsprechende Erläuterung zum P-Befehl).

Das Fragezeichen kann im Suchbegriff als Joker verwendet werden: Es steht für jedes beliebige Zeichen.

Vor Ausführung des F-Befehls wird automatisch die Wandlung von Assemblerbefehlen in Token vorgenommen. Das hat zur Folge, daß Zeichenfolgen, die Assembler-Befehlswörter eingebettet haben, so nicht gefunden werden. Zum Beispiel wird das Wort »STATUS« deswegen nicht gefunden, weil in diesem Wort der Befehl STA enthalten ist. Diese Eigenschaft läßt sich umgehen, indem man einen Buchstaben von STA durch den Joker ersetzt und beispielsweise nach »ST?TUS« sucht.

Beim Packen der Quelltextzeilen wird jedes überflüssige Leerzeichen entfernt. Sucht man beispielsweise nach »JSR LABEL«, so hat man deswegen Pech, weil Leerzeichen zwischen Assemblerbefehlen und Operanden stets entfernt worden sind. »JSRLABEL« als Suchwort bringt dagegen Erfolg.

Eine Wandlung von Pseudo-Befehlstexten in die entsprechenden Token wird hier nicht vorgenommen. Beim F-Befehl muß man daher die Token direkt eingeben. Statt nach ».MACRO« zu senden, muß man also <CBM>, <K> oder alternativ <SHIFT A> eingeben. In der Tabelle 2 und 3 finden Sie die Tastenkombinationen für alle Befehle.

G (obere Speichergrenze)
setzt die obere Speichergrenze für den Assembler. Syntax:

G (Adresse als Hex-Zahl)

Es ist die höchste Adresse anzugeben, die noch vom Assembler für das Speichern von Quelltext verwendet werden darf. Die Adresse darf nicht größer oder gleich $8000 sein. Voreinstellung ist G $7FFF; damit stehen die vollen 30 KByte für den Quelltext zur Verfügung. Durch G $5FFF würde beispielsweise der Speicherbereich von $6000 bis $7FFF für Objektcodes reserviert und gegen Überschreiben durch Quelltext geschützt.

I (Inhaltsverzeichnis) <F7>
gibt das Inhaltsverzeichnis der sich gerade im Laufwerk mit der Nummer 8 befindlichen Diskette aus. Durch Drücken der SPACE-Taste kann die Ausgabe abgebrochen werden. Dieser Befehl wird auch durch <F7> aufgerufen.

L (Load) <F5>
lädt einen Quelltext in den Speicher. Syntax:

L"Filename", Gerätenummer

Der Quelltext wird immer an die Adresse geladen, die in $2B/$2C (43/44) im Low-/High-Byte-Format steht. Dies wird im Regelfall der Basic-Start $0801 (2049 dez.) sein. Dieser Befehl wird auch durch <F5> aufgerufen. Wählen Sie dagegen den Befehl

LOAD "Filename",Gerätenummer

so wird die Angabe der Sekundäradresse 1 automatisch angenommen und das Programm absolut – also an die auf Diskette angegebene Startadresse – geladen. Eine weitere interessante Möglichkeit, Quelltexte aus dem angezeigten Directory zu laden, ist die Tastenkombination <SHIFT RUN/STOP>. Es wird nach dem Laden sofort mit der Assemblierung begonnen. Bei Verwendung von Speeddos, wozu eine bedingte Übereinstimmung der Funktions-

Token	Befehl	Taste 1	Taste 2
$A0	.CALL	<SHIFT SPACE>	
$A1	.MACRO	<CBM K>	<SHIFT A>
$A2	.ENDMACRO	<CBM I>	<SHIFT B>
$A3	.GLOBAL	<CBM T>	<SHIFT C>
$A4	.EQUATE	<CBM @>	<SHIFT D>
$A5	.BYTE	<CBM G>	<SHIFT E>
$A6	.WORD	<CBM +>	<SHIFT F>
$A7	.DS	<CBM M>	<SHIFT G>
$A8	.EXT	<CBM £>	<SHIFT H>
$A9	.OBJECT	<SHIFT £>	<SHIFT I>
$AA	.BASE	<CBM N>	<SHIFT J>
$AB	.CODE	<CBM Q>	<SHIFT K>
$AC	.ON	<CBM D>	<SHIFT L>
$AD	.GOTO	<CBM Z>	<SHIFT M>
$AE	.IF	<CBM S>	<SHIFT N>
$AF	.ELSE	<CBM P>	<SHIFT O>
$F0	.ENDIF	<CBM A>	<SHIFT P>
$F1	.SYMBOLS	<CBM E>	<SHIFT Q>
$F2	.LISTING	<CBM R>	<SHIFT R>
$F3	.END	<CBM W>	<SHIFT S>
$F4	.STOP	<CBM H>	<SHIFT T>
$F5	.PAGE	<CBM J>	<SHIFT U>
$F6	.NOCODE	<CBM L>	<SHIFT V>
$F7	.START	<CBM Y>	<SHIFT W>
$F8	.NOEXP	<CBM U>	<SHIFT X>

Tabelle 2. Token-Tabelle für Pseudo-Befehle

Token	Befehl	Taste	Token	Befehl	Taste
$C0	CPX	<SHIFT •>	$DC	TXS	<CBM ->
$C1	CPY	<SHIFT A>	$DD	PHP	<SHIFT ->
$C2	LDX	<SHIFT B>	$DE	PLP	<SHIFT !>
$C3	LDY	<SHIFT C>	$DF	PHA	<CBM •>
$C4	CMP	<SHIFT D>	$E0	PLA	<SHIFT SPACE>
$C5	ADC	<SHIFT E>	$E1	BRK	<CBM K>
$C6	AND	<SHIFT F>	$E2	RTI	<CBM I>
$C7	DEC	<SHIFT G>	$E3	RTS	<CBM T>
$C8	EOR	<SHIFT H>	$E4	NOP	<CBM @>
$C9	INC	<SHIFT I>	$E5	CLC	<CBM G>
$CA	LDA	<SHIFT J>	$E6	SEC	<CBM +>
$CB	ASL	<SHIFT K>	$E7	CLI	<CBM M>
$CC	BIT	<SHIFT L>	$E8	SEI	<CBM £>
$CD	LSR	<SHIFT M>	$E9	CLV	<SHIFT £>
$CE	ORA	<SHIFT N>	$EA	CLD	<CBM N>
$CF	ROL	<SHIFT O>	$EB	SED	<CBM Q>
$D0	ROR	<SHIFT P>	$EC	DEY	<CBM D>
$D1	SBC	<SHIFT Q>	$ED	INY	<CBM Z>
$D2	STA	<SHIFT R>	$EE	DEX	<CBM S>
$D3	STX	<SHIFT S>	$EF	INX	<CBM P>
$D4	STY	<SHIFT T>	$F0	BPL	<CBM A>
$D5	JMP	<SHIFT U>	$F1	BMI	<CBM E>
$D6	JSR	<SHIFT V>	$F2	BVC	<CBM R>
$D7	TXA	<SHIFT W>	$F3	BVS	<CBM W>
$D8	TAX	<SHIFT X>	$F4	BCC	<CBM H>
$D9	TYA	<SHIFT Y>	$F5	BCS	<CBM J>
$DA	TAY	<SHIFT Z>	$F6	BNE	<CBM L>
$DB	TSX	<SHIFT +>	$F7	BEQ	<CBM Y>

Tabelle 3. Token-Tabelle für Assembler-Befehle

tastenbelegung existiert, kann die Angabe der Laufwerksnummer sowie der Doppelpunkt hinter dem Filenamen beim Laden aus dem Directory entfallen.

M (Merge)
lädt einen Quelltext hinter den bereits im Speicher befindlichen. Syntax:

M "Filename"

Dieser Befehl wird dazu benutzt, um mehrere Quelltexte, die einzeln eingegeben wurden, von Diskette nachzuladen und zu verketten. Unter besonderen Bedingungen ist es realisierbar, mehrere unterschiedliche Quelltexte nacheinander mit dem M-Befehl einzuladen und dann in einem Durchgang zu assemblieren. Zu diesen Bedingungen zählt, daß die Speicherbereiche für die Objektcodes sich nicht überschneiden, falls direkt in den Speicher assembliert wird, und daß die Symbole in einem Quelltext nicht in einem weiteren Quelltext anders definiert werden.

N (Number)
ermöglicht ein neues Durchnumerieren von Quelltextzeilen. Syntax:

N P, Startnummer, Schrittweite

Zum Beispiel numeriert N0, 100, 10 den gesamten Quelltext neu, beginnend mit den Zeilennummern 100, 110, 120 etc. Der Paramter P hinter »N« dient wie beim Find-Befehl zur Angabe der Page. Siehe hierzu auch den P-Befehl.

Achtung: Die angesprungenen Zeilennummern bei den Befehlen .ON und .GOTO werden nicht berücksichtigt und müssen von Hand angepaßt werden! Eine solche Einschränkung wurde in Kauf genommen, damit das gesamte Programm nicht zu lang wird, um es in einem 2764-EPROM unterzubringen.

O (Old) <F6>
holt nach einem Reset oder C-Befehl den alten Quelltext zurück. Dabei wird automatisch der B-Befehl zur Anzeige der Speicherkonfiguration aufgerufen. Der O-Befehl läßt sich auch durch <F6> aufrufen.

P (Page)
setzt einen Arbeitsbereich (Page, siehe Find- und Number-Befehl). Syntax:

P N, Zeile 1, Zeile 2

Der Parameter »N« (Nummer der Page) liegt zwischen 0 und 15. »Zeile 1« gibt die erste Zeile der Page an und »Zeile 2« die letzte. Zum Beispiel setzt P1, 0, 99 die Page 1 auf den Zeilenbereich 0 bis 99. Direkt nach Initialisierung des Assemblers ist die Page 0 auf den Zeilenbereich 0 bis 65535 gesetzt, sie umfaßt also den gesamten Quelltext. Man sollte Page 0 daher zweckmäßigerweise nicht umdefinieren.

Eine Page wird auf den Bildschirm ausgegeben, indem man die <←>-Taste drückt, darauf die Page-Nummer eingibt und dann <RETURN> drückt. Auf einzelne Pages greifen die Befehle »F, N und R« zu. Gibt man dort statt einer 0 eine andere Nummer von 1 bis 15 ein, so bezieht sich der Befehl nicht auf den gesamten Quelltext, sondern eben nur auf die angegebene Page.

Q (Ausgabe der Page-Definitionen)
gibt die Definitionen aller 16 Pages auf den Bildschirm aus. Hieraus kann man ersehen, welche Pages bereits definiert wurden, um nicht ungewollt eine Page umzudefinieren.

R (Replace)
dient zum Suchen und Ersetzen von Zeichenfolgen im Quelltext. Syntax:

R P, "Ersatzwort", "Suchwort"

Der R-Befehl ersetzt »Suchwort« durch »Ersatzwort« in der angegebenen Page (anstelle der 0 kann auch eine andere Nummer von 1 bis 15 eingegeben werden). In »Suchwort« ist das Fragezeichen als Joker erlaubt. Da »Ersatzwort« auch leer sein darf, lassen sich mit diesem Befehl auch Zeichenketten löschen. Im übrigen gelten beim R-Befehl dieselben Einschränkungen wie beim F-Befehl.

S (Save)
dient zum Speichern von Quelltext auf Diskette. Syntax:

S "Filename"

Der gesamte Quelltext wird in ein PRG-File auf Diskette geschrieben. Anmerkung: Falls man zuvor den Basic-Befehl LOAD zum Laden eines Monitors verwendet hat und dadurch der Quelltextende-Zeiger verbogen wurde, sollte man vor Eingeben des S-Befehls den O-Befehl aufrufen. Dieser setzt den Quelltext-Ende-Zeiger wieder richtig.

T (Tabulatoreinstellung)
setzt die Tabulatoren 0 beziehungsweise 1 (X) auf neue Werte. Syntax:

T X, Wert

Die Tabulatoren nehmen Einfluß auf das Format, in dem Quelltextzeilen gelistet werden. Tabulator 0 bestimmt, wieviele Spalten für Labels belegt werden, und Tabulator 1 gibt an, ab welcher Spalte die Kommentare am Ende einer Zeile gelistet werden. Als erste Position innerhalb einer Quelltextzeile gilt dabei die Spalte unmittelbar hinter der Zeilennummer. Voreingestellt sind: T0,10 und T1,24.

V (Verify)
entspricht dem Basic-Befehl VERIFY. Syntax:

V "Filename"

X (Assemblieren) <F3>
startet die Assemblierung. Anstelle von »X« ist auch der Basic-Befehl RUN verwendbar. Dieser Befehl wird auch durch <F3> abgerufen.

Y (Ausgabe der Symboltabelle)
gibt nach beendeter Assemblierung die Symboltabelle auf dem Bildschirm aus. Eine Ausgabe auf den Drucker wird erreicht durch:

OPEN1,4,7: CMD1 und anschließenden Y-Befehl.

@ (Disk-Status und Floppy-Befehle)
zeigt den Status des Diskettenlaufwerks an und leitet Diskettenbefehle (I, R, N, V) ein.

Speeddos-kompatible Funktionstastenbelegung

Auf den Funktionstasten <F1>, <F3>, <F5>, <F7> liegen die Editor-Befehle »E, X, L und I«. Man beachte die Übereinstimmung zu der Speeddos-Belegung LIST, RUN, LOAD und Directory. Auf den Funktionstasten <F2>, <F4>, <F6> und <F8> liegen die Editor-Befehle »B (Speicheranzeige), Y (Ausgabe der Symboltabelle), O (Old) und @(Disk-Status)«.

Die Funktionstastenabfrage wurde in den Interrupt eingebunden. Das hat zur Folge, daß Programme, die mit Interrupts arbeiten, diese Abfrage ausschalten. Man muß nur dafür sorgen, daß nach Beendigung eines solchen Programms der Interrupt-Vektor in $0314/$0315 wieder auf den Interrupt zur Funktionstastenabfrage gesetzt wird. Dies kann auch mit den Tasten <RUN/STOP RESTORE> geschehen. Die eleganteste Lösung dafür ist, zu Beginn des Programms den alten Interruptvektor zu retten und am Ende wiederherzustellen. Es geht aber auch so, daß man am Ende der Interrupt-Benutzung den Vektor statt auf $EA31 direkt auf die entsprechende Adresse setzt, welche der Einsprungübersicht (Tabelle 4) zu entnehmen ist.

Adresse	Inhalt
$8000	RESET-Vektor, zeigt auf Kaltstart $849D
$8002	NMI-Vektor, zeigt auf NMI-Routine $9FAE
$8004	ROM-Kennung »CBM80«
$8009	INIT-Vektor, durch JMP ($8009) Initialisierung
$800B	Tabelle der Mnemonic-Texte (CPX,...,BEQ)
$80B3	Tabelle der zu den Mnemonics gehörigen Codes
$80EB	diverse Tabellen zur Mnemonic - Code-Wandlung
$8110	Tabelle der binären Operatoren »+, -, *, /, ↑, A, O, >, =, <
$811A	Texte der Assembler-Fehlermeldungen
$81E5	Adreßtabelle der einzelnen Fehlermeldungs-Texte
$8207	Texte der Pseudo-Befehle CALL bis NOEXP
$827A	Vektortabelle für Einsprünge in Pseudo-Routinen
$82AC	Tabelle der Editorkommandos M, V,...,X, Y
$82C1	Vektortabelle für Einsprünge in Editor-Routinen
$82EB	Einschaltmeldung GIGA-ASS
$832C	Texte zur Benutzerführung
$848C	Prompt »GIGA-ASS READY«
$849D	Kaltstart, wird bei RESET aufgerufen
$84AA	Initialisierung des Assemblers, beginnt mit SEI
$84AB	ROM-Bereich $8000 in RAM $8000 umkopieren
$84C4	Interrupt-Vektor auf Funktionstastenabfrage setzen
$84CF	Basic-Ende auf $8000 setzen
$84D9	Basic initialisieren
$84DF	Basic-Vektoren ab $0300 umlenken
$8507	Tabulator-Initialisierung T0, 10 und T1, 24
$8511	Page 0 auf Zeile 0 bis 65535 setzen
$8519	Einschaltmeldung ausgeben
$8537	Ausgabe des Prompts
$8544	Warmstart-Einsprung
$854A	Abschluß einer Merge-Operation
$8557	Fehlermeldung ausgeben
$85C1	Auswerten von Ausdrücken
$85F4	Erkennen der Einleitsymbole »%, $, -, !, +, >, <
$8623	Einlesen von Binärzahlen
$8647	Erkennen von !N!
$8658	Fehlermeldung beim Auswerten von Ausdrücken
$865D	Erkennen der binären Operatoren von $8110
$868E	Low-/High-Byte Bildung
$86A6	Bestimmen der Adressierungsart eines Befehls
$8710	Erkennen auf indizierte Befehle
$8781	Berechnen des Maschinencodes für ein Mnemonic
$882E	Auswerten von Branch-Befehlen (bei relativen Sprüngen)
$8870	Einlesen von Hexadezimalzahlen
$88B2	Beginn der Symboltabellen-Verwaltung
$88BA	Holen eines Symbols
$88C8	Illegales Symbol erkannt
$88DB	Erkennen des Trennzeichens < -- >
$88E6	Sucht, ob Symbol schon in Tabelle eingetragen ist
$8955	Symbol nicht gefunden - sieht auf dem Stack nach
$897D	Rücksprungadresse auf Stack ergab »Makro undefiniert«
$8985	Bei der Auswertung von Ausdrücken Symbol undefiniert
$89A4	Ansonsten wird neues Symbol eingetragen
$89E6	Symbol gefunden
$8A0F	Wert des Symbols in Fließkomma-Akku schreiben
$8A19	Symboltabelle voll!
$8A25	Holt neue Quelltextzeile
$8A4C	Ende eines Pass
$8BC6	Ordnungszahl zurücksetzen
$8BD2	Direkt-Befehl auswerten
$8BFE	RUN beziehungsweise X: Start der Assemblierung
$8C24	Startet Uhr (Assemblierungsdauer)
$8C72	Holt Adressen aller Makro-Definition in Symboltabelle
$8CC0	Nächsten Pass einleiten
$8CDC	Einsprungpunkt: nächste Quelltextzeile assemblieren
$8DDC	Pseudo-Befehl-Verteiler
$8E0A	.GLOBAL
$8E19	.EQUATE
$8E6A	.BASE
$8E79	.CODE
$8E9B	.BYTE
$8EC1	.TEXT
$8F2F	.WORD
$8F9F	.DS
$8FE1	.OBJECT
$903F	.END
$9054	.MACRO überspringt Makro-Definition
$907A	.CALL springt in Makro-Definition
$917B	.ON
$9194	.GOTO
$919A	.IF
$91A7	.ELSE
$91CD	.ENDIF
$91E5	Legt Maschinencode für eine Quelltextzeile ab
$9217	Erhöht Adreßpegel
$924C	Holt nur Zahl aus Quelltext
$925A	Holt Zahl oder Character aus Quelltext
$926A	Holt Character (String der Länge 1) aus Quelltext
$9281	Erkennt auf Token
$92E0	Gibt Uhrzeit aus (nach erfolgter Assemblierung)
$9332	.LISTING
$936A	.PAGE
$9379	Gibt Hexadezimalzahl aus
$9390	Listet Maschinencodes
$93CD	Listet Quelltextzeile, falls .LISTING aktiv
$93EB	Prüft auf Seitenwechsel bei .PAGE
$9422	.SYMBOLS
$9442	Am Ende der Assemblierung Symboltabelle ausgeben
$946C	.STOP
$947C	Blank-Ausgabe
$949F	.NOCODE
$94A6	.START
$94B2	.NOEXP
$94BF	Warmstart des Editors, holt Editor-Befehl
$953E	Listet Quelltextzeile nach Eingabe
$956D	Fügt Pseudo-Token-Code ein
$95C0	Packt und wandelt eingegebene Zeile
$96E8	Gibt Zeilennummer im Auto-Modus vor
$9749	OFF: schaltet Auto-Modus aus
$974F	A-Befehl
$9773	Holt eingegebenen Zeilen-Bereich
$97D8	D-Befehl
$9874	E-Befehl
$988E	P-Befehl setzt Page-Parameter
$98BD	Holt Page-Parameter
$98E2	Gibt Page auf Bildschirm aus
$98F3	N-Befehl
$995B	M-Befehl
$997D	Holt Filenamen
$9990	V-Befel
$9993	L-Befehl
$99A0	S-Befehl
$99A9	F-Befehl
$9A18	Holt Zeilennummer aus Quelltext
$9A23	Gibt Zeilennummer rechtsbündig aus
$9A44	Gibt eine Quelltextzeile formatiert aus
$9B13	SPACE hält Listig an
$9B33	R-Befehl
$9C4A	I-Befehl
$9CBE	@-Befehl
$9D0A	Y-Befehl
$9E13	O-Befehl
$9E2D	B-Befehl
$9EED	G-Befehl
$9F08	T-Befehl
$9F24	Q-Befehl
$9F71	IRQ-Routine: hierauf muß der IRQ-Vektor zeigen
$9FA6	Funktionstastenbelegung »E, X, L, I, B, Y, 0, @«
$9FAE	NMI-Routine
$9FFF	Letztes Byte. Der Code belegt die vollen 8 KByte!

Tabelle 4. Einsprungadressen von Giga-Ass. Wollen Sie Giga-Ass für spezielle Zwecke umschreiben oder erweitern, so finden Sie hier wichtige Basisinformationen.

Nun soll an einem konkreten Beispiel gezeigt werden, wie man einen Quelltext eingibt.

Geben Sie A 100, 10 und dann das folgende Programm ein (ohne die Zeilennummern, diese werden durch den Auto-Befehl ja vorgegeben). Achten Sie dabei darauf, daß <SPACE> bedeutet: Einmal die SPACE-Taste drücken.

```
100.BASE $6000
110.START $6000
120.MACRO PRINT <SHIFT SPACE> TEXT
130 <SPACE> LDA #< (TEXT)
140 <SPACE> LDY #> (TEXT)
150 <SPACE> JSR $AB1E
160.ENDMACRO
170 <SPACE> PRINT <SPACE> MESSAGE
180 <SPACE> RTS
190MESSAGE <SPACE> .TEXT"<beliebige Mitteilung>"
```

Nun erscheint noch die Zeilennummer 200. Diese wird aber nicht mehr benötigt, wir drücken daher <SHIFT RETURN> und noch einmal <RETURN>. Mit <F1> kann man das Programm wieder ansehen. Es besteht im wesentlichen aus dem Makro »PRINT«, das eine beliebige Mitteilung auf dem Bildschirm ausgibt. Der verwendete Parameter »TEXT« ist eine Adresse, die auf den Textanfang verweist, in unserem Fall symbolisch angegeben durch das Label »MESSAGE«. Der Aufruf für dieses Makro steht in Zeile 170.

Der Text in Zeile 190 wird zwischen Anführungszeichen genauso eingegeben, wie man das vom Basic-Befehl PRINT her gewohnt ist (inklusive aller Cursor-Steuerzeichen). Die eigentliche Text-Ausgabe erfolgt durch einen Einsprung in die entsprechende Routine des Betriebssystems ($AB1E).

Nun startet man den Assembler entweder durch <F3>, <X> <RETURN> oder auch durch »RUN« und <RETURN>. Sobald die Assemblierung beendet ist, erscheint die Meldung:

»<SPACE> FOR .START OR <RUN/STOP>«

Folgen Sie dieser Aufforderung und drücken <SPACE>, so wird der Bildschirm gelöscht und dann der Text aus Zeile 190 ausgegeben.

Es können also Speicheradressen wie gewohnt hexadezimal mit einleitendem $-Zeichen eingegeben werden. Im Beispiel wurde die Basis-Adresse auf $6000 gesetzt. Man hätte aber auch genausogut .BASE 24576 (also dezimal) eingeben können. Dezimalzahlen werden also nicht besonders gekennzeichnet. Ferner erkennt der Assembler auch Binärzahlen, die durch »%« eingeleitet werden. So ist %1010 = $A = 10.

Sehr wichtig beim Eingeben von Quelltexten ist die Unterscheidung von <SPACE> und <SHIFT SPACE>. Im Kapitel über Makro-Programmierung wird darauf noch eingegangen. Hier noch ein wichtiger Hinweis: Es kann vorkommen, daß beim Assemblieren eine auf den ersten Blick syntaktisch korrekte Zeile mit einem Fehler quittiert wird. Schuld daran ist in vielen Fällen ein verstecktes <SHIFT SPACE>, das leider beim Original-Zeichensatz von einem <SPACE> nicht zu unterscheiden ist. Eine Hilfe für das Programmieren mit Giga-Ass wäre daher ein 2732-EPROM mit einem modfizierten Zeichensatz, bei dem <SHIFT SPACE> als kleiner Punkt (wie bei Vizawrite) auf dem Bildschirm erscheint. Dieses EPROM müssen Sie anstelle des originalen Character-ROM einsetzen.

Die Token

Jede eingegebene Zeile wird nach <RETURN> gepackt und erst dann im Speicher abgelegt. Dabei werden die Assembler-Befehle wie LDA, STA etc. (die sogenannten Mnemonics) in Token umgerechnet. Statt nun tatsächlich »LDA«, »STA« einzugeben, kann man auch gleich das Grafiksymbol eintippen, welches den gleichen Zeichencode wie das entsprechende Token besitzt. Zum Beispiel kann man statt »LDA« direkt <SHIFT J> eingeben und statt STA die Tasten <SHIFT R> drücken. In Tabelle 3 sind die Token für alle Mnemonics verzeichnet.

Die Pseudo-Befehle

Die Pseudo-Befehle sind Anweisungen an den Assembler, die den Verlauf der Assemblierung steuern. Pseudo-Befehle beginnen immer mit einem Punkt. In Giga-Ass werden auch die Pseudo-Befehle als Token gespeichert. Das hat zur Folge, daß – wie bei den Assembler-Befehlen – eine abgekürzte Eingabe möglich ist. Doch aufgrund der Tatsache, daß die Grafiksymbole für je zwei unterschiedliche Codes stehen, treten Überschneidungen mit den Abkürzungen für die Assembler-Befehle auf. Dieses Problem wurde folgendermaßen gelöst: An jeder Stelle, wo ein Befehl wie LDA eingegeben wird, kann man auch das entsprechende Token direkt als Grafiksymbol eingeben.

Für die Pseudo-Befehle wurde nun eine andere Form der Abkürzung gewählt. Vermutlich ist jedem die Methode bekannt, Basic-Befehlsworte abgekürzt einzugeben: LIST kürzt man ab durch <L> <SHIFT I> und LOAD durch <L> <SHIFT O>. Es gibt aber auch Basic-Befehlsworte, die in den ersten beiden Buchstaben gleich sind, beispielsweise READ und RESTORE. Hier muß dann <R>, <E>, <SHIFT A> beziehungsweise <R>, <E>, <SHIFT S> eingegeben werden.

Das gleiche Prinzip wurde für die Pseudo-Befehle von Giga-Ass eingesetzt: .BASE kürzt man ab durch <.> <B> <SHIFT A>, .GOTO durch <.><G> <SHIFT O>. Da sich aber die meisten Pseudo-Befehle bereits im ersten Buchstaben unterscheiden, ist es im Gegensatz zu Basic zulässig, bereits den ersten Buchstaben geSHIFTet einzugeben. So wird .MACRO abgekürzt, indem man <.><SHIFT M> eingibt.

25 Pseudo-Befehle

Bei der folgenden Aufstellung aller 25 Pseudo-Befehle wird angegeben, welches die kürzeste Abkürzung für den entsprechenden Befehl ist.

.CALL
dient zum Aufruf von Makros. Dieser Pseudo-Befehl nimmt eine besondere Stellung ein, da er in dieser Form gar nicht im Quelltext erscheint. Näheres dazu weiter unten.

.MACRO <.><SHIFT M>
leitet eine Makro-Definition ein.

.ENDMACRO <.><SHIFT E>
beendet eine Makro-Definition.

.GLOBAL <.><SHIFT G>
definiert ein globales Symbol (Label). Syntax:
.GLOBAL Symbolname = Wert
Zum Beispiel wird mit:
.GLOBAL CHROUT=$ffd2
die Kernel-Routine zur Ausgabe einzelner Zeichen mit dem symbolischen Namen CHROUT benannt. Ein Label muß dann als global definiert werden, wenn es sowohl in einem Makro als auch im Hauptprogramm verwendet werden soll.

.EQUATE <.><E> <SHIFT Q>
definiert ein lokales Symbol. Syntax:
.EQUATE Symbolname = Wert

Lokale Symbole sind für Schleifen und Sprungbefehle zu verwenden. Außerdem werden sie bei jedem Aufruf eines Makros mit Parametern implizit angelegt.

.BYTE <.> <SHIFT B>
fügt einzelne Bytewerte (Werte von $00 bis $FF) in den Quelltext ein. Mehrere Bytewerte werden duch Kommata voneinander getrennt. Die Werte sind wahlweise als dezimale, hexadezimale oder binäre Zahlen, als Symbole oder als Strings der Länge 1 einzugeben. Zum Beispiel werden durch:

```
.byte 32,$20,%100000," "
```

vier Leerzeichen mit dem ASCII-Code 32 erzeugt.

.WORD <.> <SHIFT W>
fügt Adressen (16-Bit-Werte im Bereich von $0000 bis $FFFF) in den Quelltext ein. Mehrere Adressen sind durch Kommata voneinander zu trennen. Die Adressen werden in der standardmäßigen Folge Low-, High-Byte in den Objektcode aufgenommen.

.DS <.> <SHIFT D>
DS steht für »Define Storage« und dient zur Reservierung von Speicherbereichen. Syntax:
.DS »n«

Die Anzahl »n« der zu reservierenden Bytes kann zwischen 0 und 255 liegen. Vor .DS kann natürlich ein Label stehen, um den so erzeugten Speicherbereich anzusprechen. Im Gegensatz zum DS-Befehl von anderen Assemblern, die einfach den Adreßpegel erhöhen und den Speicher undefiniert lassen, initialisiert Giga-Ass den Speicherbereich mit Null. Die so erzeugten Null-Bytes werden auch ins Übersetzungs-Protokoll mit aufgenommen.

.TEXT <.> <SHIFT T>
erlaubt das Einfügen von Texten in den Quelltext. Die einzelnen Zeichen des Textes werden als ASCII-Codes im Speicher abgelegt. Es gibt zwei Syntax-Varianten:

a) .TEXT "Zeichenfolge"

Mit abschließendem Anführungszeichen fügt man hinter das letzte ASCII-Zeichen noch ein zusätzliches Nullbyte als Begrenzer. Das hat den Vorteil, daß so definierte Texte ohne weiteres mit der PRINT-Routine $AB1E ausgegeben werden können.

b) .TEXT "Zeichenfolge

Ohne abschließendes Anführungszeichen legt man nur die Zeichenfolge - ohne Nullbyte - im Objektcode ab. Nachteil ist, daß in dieser Variante abschließende Leerzeichen nicht eingegeben werden können. In diesem Falle, muß man sich mit eventuell mehreren ».BYTE 32«-Befehlen behelfen.

Die Taste <←> wurde ausgewählt, um einen Carriage-Return-Code ($0D) zu erzeugen. Damit ist es möglich, innerhalb eines .TEXT-Befehls in die nächste Zeile zu springen. Sonst müßte man das mit .BYTE $0D-Befehlen eingeben. Ein Nachteil dabei ist, daß nun das Zeichen <←> innerhalb von .TEXT-Befehlen nicht mehr erzeugt werden kann. Es muß als ».BYTE $5F« eingegeben werden.

.OBJECT <.> <SHIFT O>
lenkt die Objektcode-Ausgabe auf Diskette um. Syntax:
.OBJECT "filename,p,w"

Mit diesem Befehl ist es möglich, direkt auf Diskette zu assemblieren. Der .OBJECT-Befehl öffnet das File mit dem Namen »filename« zum Schreiben. Sollte dieses bereits vorhanden sein, so gibt Giga-Ass die Meldung FILE EXISTS ERROR aus. War das Öffnen des Files erfolgreich, so werden als erste Bytes die Basis-Adresse als Startadresse für den Objektcode im Low-, High-Byte-Format in das File geschrieben. So wird erreicht, daß das Objektprogramm an die richtige Stelle im Speicher geladen wird.

Achtung: Wird dieser Befehl verwendet, so muß der Quelltext unbedingt als letzten Befehl einen .END-Befehl enthalten! Dieser schließt das File. Sollte man diesen Befehl vergessen haben, so ist nach Beendigung der Assemblierung vor dem nächsten Diskettenzugriff das Objekt-File mit CLOSE 14 von Hand zu schließen.

Achtung: Die Befehle .OBJECT und .START (siehe unten) schließen einander aus. Der Maschinencode steht durch .OBJECT nicht im Speicher und kann daher auch nicht direkt nach der Assemblierung gestartet werden.

.BASE <.> <B> <SHIFT A>
setzt die Basis-Adresse für den Objektcode. Syntax:
.BASE Adresse

Dieser Befehl sollte zu Beginn des Quelltextes stehen.

.CODE <.> <C> <SHIFT O>
bestimmt, in welchen Speicherbereich der Objektcode abgelegt wird. Syntax:
.CODE Adresse

Der Objektcode wird ab der angegebenen Adresse in den Speicher geschrieben. Dieser Befehl ist nur notwendig, wenn der Objektcode an einer anderen Stelle des Speichers abgelegt werden soll als derjenigen, wo er später als lauffähiges Programm liegen soll (beispielsweise wenn der gewünschte Speicherbereich derzeit von einem anderen Programm belegt ist.) Ist im Quelltext kein .CODE-Befehl vorhanden, so wird der Objektcode ab der Basis-Adresse (.BASE-Befehl) im Speicher abgelegt.

.ON
ermöglicht einen bedingten Sprung. Syntax:
.ON Ausdruck, Zeilennummer

»Ausdruck« ist ein Vergleich, der -1 für wahr und 0 für falsch liefert. Ergibt »Ausdruck« als Resultat -1 (also wahr), so erfolgt der Sprung zu der angegebenen Zeilennummer; andernfalls wird mit der Assemblierung in der nächsten Zeile fortgefahren.

Als Zeilennummer wird auch ein zu berechnender oder durch .EQUATE festgelegter Ausdruck integriert. Mit einer Kombination von .EQUATE- und .ON-Befehl lassen sich sehr leicht Assemblerschleifen realisieren. Zwei Beispiele:

a) Es soll eine Tabelle erzeugt werden, in der alle Zahlen von 0 bis 99 in ihrer numerischen Reihenfolge als Bytes eingetragen sind. Wir erreichen dies durch folgendes Programm:

```
10.BASE $6000
20.EQUATE X=0
30.BYTE X
40.EQUATE X=X+1
50.ON X <100, 30
```

b) Wir möchten alle Buchstaben von »A« bis »Z« auf den Bildschirm ausgeben, aber kein Indexregister verwenden. Als Assemblerschleife läßt sich das so programmieren:

```
10.BASE $6000
20.START $6000
30.EQUATE C=65
40 <SPACE> LDA #C
50 <SPACE> JSR $FFD2
60.EQUATE C=C+1
70.ON C< 91,40
80 <SPACE> RTS
```

.GOTO <.> <G> <SHIFT O>
erzeugt einen unbedingten Sprung. Syntax:
.GOTO Ausdruck

Der »Ausdruck« wird als Zeilennummer ausgewertet. Anschließend wird die Assemblierung bei dieser Zeilennummer fortgesetzt. In Verbindung mit dem .STOP-Befehl läßt sich ein Quelltext in einzelnen Abschnitten assemblieren, um einen Fehler einzugrenzen.

.IF <.> <SHIFT I>
.ELSE <.> <E> <SHIFT L>
.ENDIF <.> <END> <SHIFT I>

ermöglichen eine bedingte Assemblierung. Syntax:
.IF Ausdruck

Wird der »Ausdruck« hinter .IF als 0 – also falsch – ausgewertet, wird die Assemblierung hinter .ELSE fortgesetzt. Wird kein .ELSE gefunden, wird die Assemblierung hinter .ENDIF fortgesetzt.

Wurde der »Ausdruck« jedoch als -1 – also wahr – ausgewertet, so wird die Assemblierung fortgesetzt, bis auf .ELSE gestoßen wird. Darauf wird .ENDIF gesucht und dahinter die Assemblierung fortgesetzt.

.SYMBOLS <.> <SHIFT S>
Syntax:
.SYMBOLS lfn, dev, sa, "filename"

Der .SYMBOLS-Befehl bewirkt, daß am Ende des Assemblierungsvorgangs die Symboltabelle ausgegeben wird. Die Parameter hinter .SYMBOLS entsprechen denen des normalen OPEN-Befehls in Basic. Die Symboltabelle ist fest auf 40 Zeichen pro Zeile formatiert und eignet sich daher auch für die Bildschirm-Ausgabe. Die Ausgabe auf dem Bildschirm wird durch Drücken der SPACE-Taste angehalten und wieder fortgesetzt oder durch <RUN/STOP> abgebrochen.
Beispiele:
.SYMBOLS 1,3,0 gibt die Symboltabelle auf den Bildschirm aus.
.SYMBOLS 1,4,0 gibt die Symboltabelle auf den Drucker aus.
.SYMBOLS 2,8,2,"table,s,w" schreibt die Symboltabelle in ein sequentielles File mit dem Namen "table".

.LISTING <.> <SHIFT L>
Syntax:
.LISTING lfn, dev, sa, "filename"

Der .LISTING-Befehl bewirkt, daß während der Assemblierung ein Übersetzungs-Protokoll auf das angegebene Gerät ausgegeben wird. Achtung: Sollten .LISTING und .SYMBOLS zusammen verwendet werden, so müssen sich die logischen Filenummern »lfn« unterscheiden, sonst gibt der Assembler die Meldung FILE OPEN ERROR aus. Die Parameter werden wie bei .SYMBOLS gesetzt.

.END
bewirkt ein Schließen des Objekt-Files (siehe .OBJECT). Etwaiger Quelltext nach .END wird bis zum nächsten .OBJECT in den Speicher assembliert.

.STOP <.> <S> <SHIFT T>
bewirkt einen Abbruch der Assemblierung – jedoch erst im dritten Pass. Dadurch ist man sicher, daß alle Labels und Makros zur Verfügung stehen. Der Objektcode wird jedoch nur bis .STOP erzeugt.

.PAGE <.> <SHIFT P>
Syntax:
PAGE n

Dieser Befehl bewirkt eine Seitenformatierung. Nach »n« Quelltextzeilen wird bei Bildschirmausgabe der Bildschirm gelöscht, beim Drucker ein Form Feed gesendet.

.NOCODE <.> <SHIFT N>
unterdrückt das Ablegen von Objektcode im Speicher. Sinnvoll ist dies beim direkten Assemblieren auf Diskette.

.START <.> <S> <T> <SHIFT A>
Syntax:
.START Adresse

Wenn man diesen Befehl eingegeben hat, so wird nach beendeter Assemblierung zur angegebenen Adresse gesprungen, nachdem auf die SPACE-Taste gewartet und der Bildschirm gelöscht wurde. Das aufgerufene Programm muß mit RTS enden, damit ein Rücksprung in den Assembler erfolgt. Diesen Befehl kann man in den Quellcode einbinden, um einen Probedurchlauf zu erzeugen.

.NOEXP <.> <N> <O> <SHIFT E>
unterdrückt die Makro-Expansion im Übersetzungs-Protokoll, was bedeutet, daß der innerhalb von Makroaufrufen erzeugte Objektcode nicht gelistet wird. Besonders bei mehrfach aufgerufenen Makros dient dies zur Übersichtlichkeit im Protokoll.

Die Token für die Pseudo-Befehle sind in Tabelle 2 angegeben. Angewendet werden diese Token bei den Editor-Befehlen F und R. Soll beispielsweise nach jedem Auftreten des Pseudo-Befehls .MACRO gesucht werden, so lautet der Befehl: F0,"<CBM K>".

Für das Eingeben von Quelltextzeilen gilt folgende Sonderregelung:

In der ersten Spalte hinter der Zeilennummer dürfen die Pseudo-Befehle auch als Token eingegeben werden. Steht jedoch in der betreffenden Zeile ein Label oder rückt man den Pseudo-Befehl durch ein Leerzeichen ein, so werden nur noch Token für Assembler-Befehle akzeptiert.

Diese Sonderregelung ist notwendig wegen der Überlappung bei den Grafiksymbolen, die für je zwei unterschiedliche Codes stehen. Dabei müssen die Token für Pseudo-Befehle stets am Anfang einer Zeile stehen.

In der ersten Spalte hinter der Zeilennummer dürfen statt den normalen Token auch die sogenannten Zusatz-Token eingegeben werden. Bei diesen Token handelt ich ausschließlich um geSHIFTete Tasten (Tabelle 2, Taste 2).

Die Makro-Programmierung

Makros sind meist kürzere Befehlsfolgen, die im Quelltext häufiger vorkommen und deshalb zu einer Einheit zusammengefaßt werden. Zu jedem Makro gehört ein Name, mit dem es aufgerufen wird. An jedes Makro lassen sich beliebig viele Parameter übergeben, deren aktueller Wert dann bei der Assemblierung eingesetzt wird. Makro-Definitionen sind an beliebiger Stelle im Quelltext erlaubt. Alle Makro-Namen sind global in dem Sinne, daß innerhalb eines Makros jedes andere aufgerufen werden kann. Alle Parameter und Makro-internen Label sind lokal. Das heißt, verschiedene Makros dürfen durchaus Label beziehungsweise Parameter gleichen Namens verwenden.

Sehen wir uns die prinzipielle Form einer Makro-Definition anhand eines Beispiels an:

```
100.BASE $6000
110.START $6000
120.MACRO POKE <SHIFT SPACE> ADR, VAL
130.<SPACE> LDA #VAL
140 <SPACE> STA ADR
150.ENDMACRO
160 <SPACE> POKE <SPACE> 53280,0
170 <SPACE> RTS
```

Der .MACRO-Befehl wird gefolgt vom Makro-Namen und einer Parameterliste, wobei dazwischen ein »SHIFT SPACE« stehen muß. Man nennt diese Parameter auch »formale Parameter«. Mehrere solcher Parameter werden durch Kommata getrennt. Die Reihenfolge der formalen Parameter ist willkürlich und bleibt Ihnen überlassen, weil es sich um eine Definition handelt. Nur müssen Sie sich später bei der Verwendung des Makros auch daran halten. In unserem Beispiel ist der Makro-Name »POKE« und die Parameter sind »ADR VAL«.

Die Quelltextzeilen, die direkt auf den .MACRO-Befehl folgen, definieren die Befehlsfolge, die bei jedem Makro-Aufruf im Objekt-Programm abgelegt wird. Innerhalb dieser Befehlsfolge tritt jeder Parameter an beliebigen Stellen

auf, und zwar beliebig oft, jedoch mindestens einmal (sonst wäre der Parameter ja sinnlos). Abgeschlossen wird die Befehlsfolge durch den Pseudo-Befehl .ENDMACRO.

Wir betrachten den Makro-Aufruf aus Zeile 160.

```
POKE <SPACE> 53280,0
```

Auf den Makro-Namen »POKE« folgt die Liste der sogenannten »aktuellen Parameter« 53280 und 0. Zwischen Makro-Name und den Parametern muß im Gegensatz zu Assembler-Befehlen und deren Operanden ein Leerzeichen stehen. Stößt der Assembler auf diesen Makro-Aufruf, so tut er folgendes: Er merkt sich die aktuellen Parameter 53280 und 0. Darauf springt er an die Stelle des Quelltextes, wo sich die Definition des POKE-Makros befindet. Nun setzt er die aktuellen Parameter 53280 und 0 für die formalen Parameter »ADR« und »VAL« ein: »ADR« wird auf 53280 gesetzt und »VAL« auf 0. Damit ergibt sich die Befehlsfolge: LDA #0, STA 53280, die bekanntlich die Rahmenfarbe auf Schwarz setzt. Diese Befehlsfolge wird im Objekt-Programm abgelegt. Danach stößt der Assembler auf .ENDMACRO und springt im Quelltext an die Stelle hinter dem Makro-Aufruf zurück.

So »schachtelt« man Makros

Dazu ein Beispiel (das POKE-Makro sei nach wie vor definiert):

```
200.MACRO RAM
210 <SPACE> POKE 1,$36
220.ENDMACR
300.MACRO ROM
310 <SPACE> POKE 1,$37
320.ENDMACRO
```

Das RAM-Makro blendet das Basic-ROM aus; an dessen Stelle tritt der RAM-Bereich von $A000 bis $BFFF. Das ROM-Makro macht es wieder rückgängig.

Diese Makro-Definition nennt man »geschachtelt«, weil innerhalb eines Makros ein anderes Makro aufrufen wird. Die Schachtelungstiefe ist nur begrenzt durch die Größe des Prozessor-Stack.

Zur Wahl der Makro-Namen

Der Makroname sollte nicht zu lang gewählt werden, denn bei jedem Auruf muß man ihn so eintippen, wie man ihn definiert hat. Wichtiger ist jedoch, daß der Makro-Name nicht mit einem Mnemonic (also LDA, STA etc.) beginnt. Das ist zwar nicht verboten, hat aber trotzdem einen guten Grund. Ein Beispiel: Nehmen wir an, wir haben ein Makro »STATUS« definiert, das den Disk-Status ausgibt. Geben wir nun das Wort »STATUS« in eine Quelltextzeile ein, so vermag der Assembler prinzipiell nicht zu unterscheiden, ob es sich nun um ein Makro mit dem Namen »STATUS« handelt oder um den Assembler-Befehl STA mit »TUS« als symbolischem Operanden.

Sicher erkennen Sie das Problem: Soll die Tokenwandlung erfolgen oder nicht? Sie erfolgt, weil den Mnemonics immer der Vorrang eingeräumt wird. Trotzdem gibt es einen Weg, die Tokenwandlung zu unterdrücken: Man gibt vor dem Makro-Namen ein <SHIFT SPACE> ein. Dies ist das Token für den Pseudo-Befehl .CALL. Nur dann, wenn ein Makro-Name mit einem Mnemonic beginnt, muß es eingegeben werden, sonst ist es unnötig.

Rechnungen im Quelltext

Im Quelltext sind die vier Grundrechenarten Addition (+), Subtraktion (–), Multiplikation (*) und Division (/) sowie

auch die Potenzierung (↑) erlaubt. Daneben gibt es noch die drei Vergleichsoperatoren »<, =, >« sowie vier logische Operatoren. Dies sind zunächst die beiden Operatoren »N« für das Einerkomplement (NOT) und »-« für das Zweierkomplement sowie zwei binäre Operatoren: »A« für das logische »AND« und »O«, das logische »OR«.

Für die Operatoren »N, A und O« gibt es eine Sonderregelung: Um diese Zeichen von Symbol-Zeichen zu trennen, müssen sie in Ausrufezeichen eingeschlossen werden. Bei NOT ist das obligatorisch, es muß immer als »!N!« geschrieben werden. Für AND und OR gilt: Nur wenn die Operanden Zahlen sind, kommt man ohne Ausrufungszeichen aus, beispielsweise ist:

1 A 1 = 1. Eine logische UND-Verknüpfung zwischen den beiden Symbolen »X« und »Y« erreicht man durch: X !A! Y.

Die Vergleichsoperatoren liefern als Wahrheitswerte -1 für wahr und 0 für falsch. Eine logische Negation (NOT) erreicht man durch den N-Operator. Es ist: !N! -1 = 0 und !N! 0 = -1. Damit lassen sich auch die Operatoren »< >«, »< =« und »= >« (ungleich, kleiner und größer gleich), die nicht direkt zur Verfügung stehen, realisieren: Es ist:

A< >B = !N! A = B
A < = B = !N! A > B
A > = B = !N! A < B

Beim Rechnen mit den Wahrheitswerten muß man darauf achten, daß man -1 und 0 zwar als Operanden für .IF und .ON verwenden, aber -1 nicht einem Symbol zugeweisen darf. Um Wahrheitswerte in Variablen zu speichern, muß man ins Zweierkomplement wandeln und anstelle mit -1 und 0 mit +1 und 0 rechnen. Zum Beispiel wird durch den Befehl .EQUATE C = -(A=B) das Symbol C auf 1 gesetzt, falls A=B gilt, und sonst auf 0.

Das Adreßpegelsymbol »*«

Das Symbol »*« liefert als Wert die Adresse zurück, an welcher der Assembler sich im Augenblick befindet. Diese Adresse nennt man auch den »Adreßpegel«. Sie gibt an, an welcher Adresse der Assembler das nächste Byte im Speicher ablegt.

Zur Verdeutlichung ein Beispiel. Geben Sie einmal folgendes Program ein:

```
10.BASE $6000
20.START $6000
30 <SPACE> LDX #0
40 MARKE TXA
50 <SPACE> STA $400,X
60 <SPACE> INX
70 <SPACE> BNE MARKE
80 <SPACE> RTS
```

Das Programm schreibt alle 256 Zeichen des Zeichensatzes ins obere Viertel des Bildschirms. Betrachten wir nun den Wert des Symbols »*« im Verlauf der Assemblierung:

```
30 *=$6000
40 *=$6002, MARKE = *
50 *=$6003
60 *=$6006
70 *=$6007
80 *=$6009
```

Man kann nun die hauptsächliche Verwendung des Symbols »*« erklären: Es dient zur Einsparung von Labeln bei relativen Sprüngen. Im Beispiel hat das Label »MARKE« den Wert $6002. Zugegriffen wird auf das Label in Zeile 70. Dort gilt * =$6007. Aber es ist auch *-5 = MARKE = $6002. Also hindert uns nichts daran, in Zeile 70 das Label »MARKE« durch den Ausdruck »*-5« zu ersetzen! Damit ist das Label »MARKE« in Zeile 40 überflüssig geworden; man kann es bedenkenlos wieder entfernen.

Vorteilhaft wird diese Variante in längeren Programmen, wo man es irgendwann leid ist, sich für jede Schleife einen neuen Label-Namen zu überlegen. Allerdings hat die Sache einen Haken: Man muß anfangen zu zählen, und zwar die Anzahl der Bytes, die durch den relativen Sprung übersprungen werden sollen. Hierfür gibt es folgende Regeln:

1. Bei Rückwärts-Sprüngen zählt man rückwärts die Anzahl der Bytes ab der Zeile direkt vor dem Sprungbefehl bis zur Zeile, in der das Sprungziel steht. In jeder Zeile zählt man dabei die Anzahl der Bytes, die der Befehl im Objekt-Programm belegen wird. Aufpassen müssen Sie vor allem bei der Unterscheidung von 2- und 3-Byte-Befehlen!

2. Bei Vorwärts-Sprüngen wird die Zeile, in der der Sprungbefehl steht, mitgezählt. Man zählt vorwärts bis zur Zeile unmittelar vor dem Sprungziel.

3. Besondere Vorsicht ist geboten, wenn man »*« als Parameter bei Makro-Aufrufen verwendet! Es wird dann der Adreßpegel im ersten Byte des expandierten Makros genommen. Hierzu ein Beispiel:

```
10.BASE $6000
20.START $6000
30.MACRO EQB <SHIFT SPACE> A,B
40 <SPACE> LDA A
50 <SPACE>  BEQ B
60.ENDMACRO
70 <SPACE> LDA # 0
80 <SPACE> STA 198
90 <SPACE> EQB <SPACE> 198,*
99 <SPACE> RTS
```

Dieses Programm wartet darauf, daß die Anzahl der gedrückten Tasten ungleich 0 wird. Der EQB-Makro-Aufruf entspricht der Befehlsfolge:

LDA 198, BEQ *-2.

Das »&«-Symbol

Schließlich kommen wir noch zum Symbol »&«. Es gibt an, daß die darauf folgende Zero-Page-Adresse indirekt angesprochen wird. Damit wird die normale Notation, bei der die Zero-Page-Adresse eingeklammert wird, abgekürzt.

So kann man z.B. statt LDA (ADR,X) schreiben LDA &ADR,X und STA (ADR),Y abkürzen durch STA &ADR,Y.

Fehlermeldungen von Giga-Ass

Tritt während der Assemblierung ein Fehler auf, so wird die Assemblierung abgebrochen und eine Fehlermeldung sowie die fehlerhafte Zeile ausgegeben. Zur Fehlerbehebung ist die Symbol-Tabelle von Nutzen. Ferner lassen sich mit dem Befehl »/«, welcher den PRINT-Befehl aus dem Basic ersetzt, einzelne Symbol-Werte auf dem Bildschirm ausgeben. Wurde der Fehler innerhalb eines Makros entdeckt, so stehen nur die Symbole innerhalb des Makros zur Verfügung. Geben Sie bitte

```
POKE $8B, 0: POKE $8C, 0
```

ein, um auf die globalen Symbole zuzugreifen.

Eigene Fehlermeldungen

SYNTAX ERROR

Eine Quelltextzeile wurde nicht richtig gebildet. Beachten Sie bitte die richtige Reihenfolge Label, Assembler-Befehl, Operanden, Kommentar.

ILLEGAL QUANTITY ERROR

Ein Wert liegt außerhalb des zulässigen Bereichs. Byte-Werte liegen im Bereich $00 bis $FF, Wort- und Symbol-Werte im Bereich $0000 bis $FFFF. Alles, was darüber hinausgeht, führt zu dieser Fehlermeldung.

TERM EVALUATION ERROR
Ein Ausdruck konnte nicht berechnet werden. Beachten Sie hierzu das Kapitel über Berechnungen im Quelltext.

TOKEN ERROR
Zulässige Token liegen im Bereich $A0 bis $B8 für die Pseudos und $C0 bis $F7 für die Mnemonics. Treten andere Byte-Werte größer oder gleich $80 im Quelltext auf, so tritt dieser Fehler auf. Ausnahme: In der ersten Zeile steht ein SYS-Befehl mit dem Token $9E. In diesem Fall wird der Programmtext nicht assembliert, sondern als Maschinenprogramm betrachtet und mit dem SYS-Befehl gestartet. Damit lassen sich auch innerhalb des Assemblers Maschinenprogramme einladen und starten.

INDEXING ERROR
Ein Assembler-Befehl existiert nicht in der Kombination von gewählter Adressierungsart und gewähltem Register.

LINE FORMAT ERROR
Das Zeilenformat für Quelltextzeilen wurde nicht eingehalten. Beachten Sie, daß Kommentare am Schluß der Zeile mit Semikolon abgetrennt werden müssen.

ADRESSING ERROR
Ein Assembler-Befehl existiert nicht in der gewählten Adressierungsart.

BRANCH ERROR
Ein relativer Sprung führt über eine zu große Distanz (größer als 128 Byte). Ersetzen Sie diesen Branch-Befehl durch einen »Long Branch« mit Hilfe eines JMP-Befehls.

UNDEFINED SYMBOL ERROR
Ein Symbol wurde nicht vor seiner erstmaligen Verwendung definiert. Undefinierte Symbole haben den Wert $FFFF; dieser Wert darf also nie einem Symbol zugewiesen werden. Der Fehler tritt auch dann auf, wenn innerhab eines Makros ein lokales Label außerhalb der Makro-Definition angesprochen wird. Definieren Sie hierzu dieses Label als global vor der Definition des Makros.

ILLEGAL SYMBOL ERROR
Dieser Fehler tritt dann auf, wenn ein Symbol nicht mit einem Buchstaben beginnt. Beachten Sie, daß Groß-/Kleinschreibung bei Symbolen nicht erlaubt ist. Als Trennzeichen innerhalb von Symbolen kann < ← > verwendet werden.

SYMBOL TABLE FULL ERROR
Die Symboltabelle ist mit 1170 Symbolen gefüllt, wenn dieser Fehler auftritt. Das passiert nur dann, wenn sehr viele Makros mit sehr vielen Parametern sehr oft aufgerufen werden – also praktisch nie.

NO MACRO TO CLOSE ERROR
Es kommt ein .ENDMACRO zuviel im Quelltext vor! Dieser Fehler tritt auch dann auf, falls vor einem .MACRO-Befehl ein Label steht. Dies ist so nicht erlaubt. Schreiben Sie das Label allein in eine freie Zeile direkt vor dem .MACRO-Befehl!

DOUBLE LABEL ERROR
Sie haben versucht, einen Label-Namen zweimal zu definieren. Geben Sie das zweite Mal einen anderen Namen ein. Kritisch ist es, wenn Sie innerhalb einer Assemblerschleife Label definieren. Dort ist es dann notwendig, mit dem »*«-Symbol zu arbeiten.

PARAMETER ERROR
Die Anzahl der Parameter in einem Makro-Aufruf stimmt nicht mit der Definition überein.

RETURN ERROR
Diese Fehlermeldung wird nur dann erzeugt, wenn Sie mit einem Assembler-Sprung mitten in eine Makro-Definition hineinspringen. Kontrollieren Sie die Zeilennummern in Ihren Sprungbefehlen.

UNDEFINED MACRO ERROR
Bei einem Makro-Aufruf wurde die zugehörige Makro-Definition nicht gefunden.

MACRO NOT CLOSED ERROR
zeigt einen grundlegenden Fehler an. Sie haben vergessen, Ihre Makro-Definition mit .ENDMACRO abzuschließen. Dieser Fehler tritt auch dann auf, wenn vor einem .ENDMACRO-Befehl ein Label steht. Dies ist nicht erlaubt. Schreiben Sie das Label allein in eine freie Zeile direkt vor dem .ENDMACRO-Befehl!

IF-ELSE-ENDIF ERROR
Achten Sie darauf, daß die Reihenfolge der Befehle .IF, .ELSE und .ENDIF stimmt und daß eine IF-ELSE-ENDIF-Konstruktion nicht geschachtelt werden darf.
Dieser Fehler tritt auch auf, falls vor einem .IF, .ELSE oder .ENDIF-Befehl ein Label steht. Dies ist nicht erlaubt. Schreiben Sie das Label allein in eine freie Zeile direkt vor dem Befehl!

.BASE MISSING ERROR
Jeder Quelltext muß unbedingt eine Basis-Adresse enthalten. Definieren Sie diese am besten direkt zu Beginn des Quelltextes.

Die Synchronisation

Es gibt eine Situation, in der der Assembler überfordert ist, und zwar dann, wenn Vorwärtsreferenzen auf Symbole erfolgen. Das bedeutet, daß ein Symbol verwendet wird, bevor es definiert wurde. Dieser Fall tritt dann ein, wenn ein Assembler-Befehl auf ein Symbol zugreift, aber dieses Symbol erst später im Quelltext definiert wird. In diesem Fall mußte eine Festlegung erfolgen, daß dieses zunächst unbekannte Symbol mit $8000 initialisiert wird. Ein Assembler-Befehl, der eine Vorwärtsreferenz beinhaltet, wird also prinzipiell als 3-Byte-Befehl interpretiert. (Eine Ausnahme ist die Immediate-Adressierung. Hier ist klar, daß es sich um einen 2-Byte-Befehl handelt. Die Initialisierung erfolgt dann mit $80.) Zum Glück gibt es nur eine einzige Situation, in der dieser Fehler auftritt:

```
100.BASE $6000
110 <SPACE> LDA A
120 <SPACE> RTS
130.EQUATE A=198
```

Im ersten Pass nimmt der Assembler an, daß es sich in Zeile 110 bei LDA A um einen 3-Byte-Befehl handelt, denn er hat ja A=$8000 initialisiert. In Zeile 120 gilt dann * =$6003.

Im zweiten Pass nimmt der Assembler an, daß es sich in Zeile 120 bei LDA A um einen 2-Byte-Befehl handelt, denn er hat ja A=198, also eine Zero-Page-Adresse, in der Symboltabelle eingetragen. In Zeile 120 gilt dann * =$6002.

Diese Sitution, daß im ersten und im zweiten Pass unterschiedliche Adreßpegel ermittelt werden, nennt man einen Synchronisationsfehler. Der Assembler selbst merkt davon nichts, er erzeugt auch keine Fehlermeldung. Nur der Objectcode wird fehlerhaft, weil die Lücke von einem Byte bei der weiteren Assemblierung katastrophale Folgen hat.

Glücklicherweise tritt in Giga-Ass ein Synchronisationsfehler nur bei Vorwärtsreferenzen auf Zero-Page-Adressen auf, also wenn eine Zero-Page-Adresse erst hinter ihrer ersten Verwendung definiert wird. Man umgeht solche Vorwärtsreferenzen dadurch, daß man ein für allemal festlegt,

daß Zero-Page-Adressen am Anfang des Quelltextes zu definieren sind. Dies sei Ihnen hiermit ans Herz gelegt und entspricht auch einem guten Programmierstil.

Kompatibilität zum Hypra-Ass

Um für Giga-Ass eine Kompatibilität zum Hypra-Ass sicherzustellen, stellen wir Ihnen das Konvertierprogramm »Hypra-Konvert« zur Verfügung. Dessen Quellcode finden Sie in Listing 2. Listing 3 beinhaltet den Objekt-Code dieses Programms als Basic-Start-Version. Eine Konvertierung ist aufgrund der Tokenverarbeitung von Giga-Ass notwendig, da Hypra-Ass alle Befehle und Mnemonics im Klartext auf Diskette speicherte.

Listing 4 (ebenfalls Quell-Code) verschiebt den Bildschirmspeicher des C64 nach $CC00, so daß Sie ein weiteres KByte für Quelltexte zur Verfügung haben.

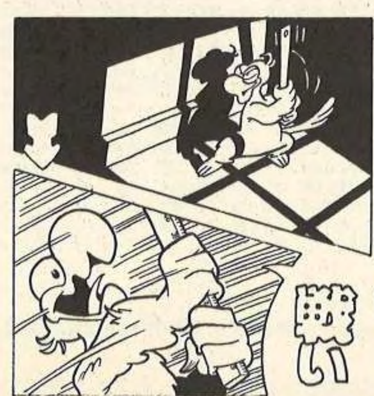
Nun bleibt uns noch, Ihnen viel Erfolg und Spaß bei Ihren weiteren Programm-Projekten mit Giga-Ass zu wünschen. Sollten Sie Anregungen oder Verbesserungsvorschläge zu diesem Spitzen-Assembler haben, können Sie uns diese gerne zukommen lassen. Besonders interessiert sind wir an weiteren Konvertierprogrammen von anderen Assemblern nach Giga-Ass (siehe Aufruf Seite 124).

(Thomas Dachsel/sk)

```
Name : giga-ass              8000 a000
----------------------------------------
8000 : 9d 84 ae 9f c3 c2 cd 38   79
8008 : 30 aa 84 43 50 58 43 50   8c
8010 : 59 4c 44 58 4c 44 59 43   7e
8018 : 4d 50 41 44 43 41 4e 44   66
8020 : 44 45 43 45 4f 52 49 4e   ca
8028 : 43 4c 44 41 41 53 4c 42   2f
8030 : 49 54 4c 53 52 4f 52 41   8c
8038 : 52 4f 4c 52 4f 52 53 42   e9
8040 : 43 53 54 41 53 54 58 53   4a
8048 : 54 59 4a 4d 50 4a 53 52   ce
8050 : 54 58 41 54 41 58 54 59   86
8058 : 41 54 41 59 54 53 58 54   29
8060 : 58 53 50 48 50 50 4c 50   d8
8068 : 50 48 41 50 4c 41 42 52   b3
8070 : 4b 52 54 49 52 54 53 4e   d4
8078 : 4f 50 43 4c 43 53 45 43   b4
8080 : 43 4c 49 53 45 49 43 4c   ea
8088 : 56 43 4c 44 53 45 44 44   14
8090 : 45 59 49 4e 59 44 45 58   1b
8098 : 49 4e 58 42 50 4c 42 4d   72
80a0 : 49 42 56 43 42 56 53 42   b1
80a8 : 43 43 42 43 53 42 4e 45   91
80b0 : 42 45 51 e4 c4 a6 a4 c5   25
80b8 : 65 25 c6 45 e6 a5 06 24   06
80c0 : 46 05 26 66 e5 85 86 84   8d
80c8 : 4c 20 8a aa 98 a8 ba 9a   0b
80d0 : 08 28 48 68 00 40 60 ea   65
80d8 : 18 38 58 78 b8 d8 f8 88   79
80e0 : c8 ca e8 10 30 50 70 90   b2
80e8 : b0 d0 f0 40 40 54 68 7b   84
80f0 : 7b 7b 28 7b 28 7b a8 00   a3
80f8 : a8 7b a8 a8 7b 3b 04 08   4f
8100 : 0c fc 10 10 14 18 04 04   ab
8108 : 02 02 02 02 03 03 02 01   1f
8110 : 2b 2d 2a 2f 5e 41 4f 3e   ec
8118 : 3d 3c 54 45 52 4d 20 45   cc
8120 : 56 41 4c 55 41 54 49 4f   4f
8128 : ce 54 4f 4b 45 ce 49 4e   ea
8130 : 44 45 58 49 4e c7 4c 49   3d
8138 : 4e 45 20 46 4f 52 4d 41   39
8140 : d4 41 44 44 52 45 53 53   92
8148 : 49 4e c7 42 52 41 4e 43   e1
8150 : c8 55 4e 44 45 46 49 4e   27
8158 : 45 44 20 53 59 4d 42 4f   d9
8160 : cc 49 4c 4c 45 47 41 4c   9a
8168 : 20 53 59 4d 42 4f cc 53   aa
8170 : 59 4d 42 4f 4c 20 54 41   84
8178 : 42 4c 45 20 46 55 4c cc   10
8180 : 4e 4f 20 4d 41 43 52 4f   3e
8188 : 20 54 4f 20 43 4c 4f 53   25
8190 : c5 44 4f 55 42 4c 45 20   d2
8198 : 4c 41 42 45 cc 50 41 52   b7
81a0 : 41 4d 45 54 45 d2 52 45   22
81a8 : 54 55 52 ce 55 4e 44 45   79
81b0 : 46 49 4e 45 44 20 4d 41   d4
81b8 : 43 52 cf 4d 41 43 52 4f   d8
81c0 : 20 4e 4f 54 20 43 4c 4f   52
81c8 : 53 45 c4 49 46 2d 45 4c   94
81d0 : 53 45 2d 45 4e 44 49 c6   74
81d8 : 2e 42 41 53 45 20 4d 49   ff
81e0 : 53 53 49 4e c7 1a 81 29   9f
81e8 : 81 2e 81 36 81 41 81 4b   66
81f0 : 81 51 81 61 81 6f 81 80   41
81f8 : 81 91 81 9d 81 a6 81 ac   02
8200 : 81 bb 81 cb 81 d8 81 43   a4
8208 : 41 4c cc 4d 41 43 52 cf   63
8210 : 45 4e 44 4d 41 43 52 cf   4e
8218 : 47 4c 4f 42 41 cc 45 51   d4
8220 : 55 41 54 c5 42 59 54 c5   af
8228 : 57 4f 52 c4 44 d3 54 45   13
8230 : 58 d4 4f 42 4a 45 43 d4   94
8238 : 42 41 53 c5 43 4f 44 c5   f4
8240 : 4f ce 47 4f 54 cf 49 c6   29
8248 : 45 4c 53 c5 45 4e 44 49   ab
8250 : c6 53 59 4d 42 4f 4c d3   37
8258 : 4c 49 53 54 49 4e c7 45   59
8260 : 4e c4 53 54 4f d0 50 41   af
8268 : 47 c5 4e 4f 43 4f 44 c5   5b
8270 : 53 54 41 52 d4 4e 4f 45   0f
8278 : 58 d0 79 90 53 90 52 91   cf
8280 : 09 8e 18 8e 9a 8e 2e 8f   9e
8288 : 9e 8f c0 8e e0 8f 69 8e   3d
8290 : 78 8e 7a 91 93 91 99 91   6f
8298 : a6 91 cc 91 21 94 31 93   0f
82a0 : 3e 90 6b 94 69 93 9e 94   6b
82a8 : a5 94 b1 94 4d 56 4c 53   f6
82b0 : 41 44 4e 45 54 50 46 52   d5
82b8 : 49 40 51 4f 42 47 43 58   7c
82c0 : 59 5a 99 8f 99 92 99 9f   72
82c8 : 99 4e 97 d7 97 f2 98 73   c3
82d0 : 98 07 9f 8d 98 a8 99 32   1f
82d8 : 9b 49 9c bd 9c 23 9f 12   7c
82e0 : 9e 2c 9e ec 9e a9 84 fd   1f
82e8 : 8b 09 9d 93 0d 20 20 2a   78
82f0 : 2a 2a 20 47 49 47 41 2d   4f
82f8 : 41 53 53 20 28 43 29 20   3d
8300 : 4d 41 52 4b 54 20 26 20   0b
8308 : 54 45 43 48 4e 49 4b 20   75
8310 : 2a 2a 2a 0d 0d 20 20 42   52
8318 : 59 20 54 48 4f 4d 41 53   ab
8320 : 20 44 41 43 48 53 45 4c   e8
8328 : 20 20 20 00 0d 41 53 53   2f
8330 : 45 4d 42 4c 59 20 42 45   60
8338 : 47 49 4e 53 0d 0d 00 0d   75
8340 : 45 4e 44 20 4f 46 20 41   ec
8348 : 53 53 45 4d 42 4c 59 0d   46
8350 : 00 50 41 47 45 20 00 3a   7b
8358 : 20 4c 49 4e 45 20 00 50   b0
8360 : 41 53 53 20 00 0d 41 53   38
8368 : 53 45 4d 42 4c 59 20 54   b2
8370 : 49 4d 45 20 55 53 45 44   43
8378 : 00 4f 42 4a 45 43 54 20   fa
8380 : 52 41 4e 47 45 20 24 00   d5
8388 : 20 2d 20 24 00 4c 49 4e   f0
8390 : 45 20 23 20 20 4c 4f 43   da
8398 : 20 20 20 20 43 4f 44 45   1f
83a0 : 20 20 20 20 20 20 20 4c   f9
83a8 : 49 4e 45 00 0d 53 59 4d   d5
83b0 : 42 4f 4c 20 54 41 42 4c   a2
83b8 : 45 3a 0d 00 0d 41 52 45   0c
83c0 : 20 59 4f 55 20 53 55 52   a2
83c8 : 45 3f 20 00 20 42 59 54   d7
83d0 : 45 53 20 55 53 45 44 20   22
83d8 : 20 20 20 20 20 00 0d 44   d3
83e0 : 49 53 4b 20 45 52 52 4f   79
83e8 : 52 3a 20 00 47 4c 4f 42   f8
83f0 : 41 4c 00 20 4d 41 43 52   ec
83f8 : 4f 00 0d 53 4f 55 52 43   65
8400 : 45 20 54 45 58 54 20 20   fc
8408 : 55 53 45 53 20 00 4f 42   86
8410 : 4a 45 43 54 20 43 4f 44   3a
8418 : 45 20 20 55 53 45 53 20   0d
8420 : 00 53 59 4d 42 4f 4c 20   da
8428 : 54 41 42 4c 45 20 55 53   88
8430 : 45 53 20 00 20 42 59 54   49
8438 : 45 53 00 0d 41 56 41 49   27
8440 : 4c 41 42 4c 45 20 4d 45   5c
8448 : 4d 4f 52 59 3a 20 24 00   32
8450 : 0d 53 4f 55 52 43 45 20   1a
8458 : 54 45 58 54 20 55 53 45   74
8460 : 53 3a 20 24 00 20 4b 29   dd
8468 : 0d 00 0d 3c 53 50 41 43   83
8470 : 45 3e 20 46 4f 52 20 2e   0a
8478 : 53 54 41 52 54 20 4f 52   b8
8480 : 20 3c 52 55 4e 2f 53 54   52
8488 : 4f 50 3e 00 0d 47 49 47   4e
8490 : 41 2d 41 53 53 20 52 45   2d
8498 : 41 44 59 0d 00 78 20 a3   7f
84a0 : fd 20 50 fd 20 15 fd 20   64
84a8 : 5b ff 78 a9 00 a2 80 86   7a
84b0 : 23 85 22 a0 00 b1 22 91   6c
84b8 : 22 c8 d0 f9 e6 23 a6 23   1a
84c0 : e0 a0 90 f1 a9 71 a2 9f   42
84c8 : 8e 15 03 8d 14 03 58 a9   61
84d0 : 00 a2 80 8e 84 02 8d 83   a9
84d8 : 02 20 53 e4 20 bf e3 a9   3f
84e0 : 44 a2 85 8e 01 03 8d 00   07
84e8 : 03 a9 bf a2 94 8e 03 03   d4
84f0 : 8d 02 03 a9 d2 a2 8b 8e   02
84f8 : 09 03 8d 08 03 a9 c1 a2   b1
8500 : 85 8e 0b 03 8d 0a 03 a9   78
8508 : 0a 8d 3e 03 a9 18 8d 3f   d9
8510 : 03 a9 ff 8d 60 03 8d 70   cf
8518 : 03 a9 eb a0 82 20 1e ab   f8
8520 : a5 37 38 e5 2b aa a5 38   3b
8528 : e5 2c 20 cd bd a9 66 a0   e9
8530 : e4 20 1e ab 20 44 a6 a9   33
8538 : 8c a0 84 20 1e ab 20 ba   6f
8540 : 94 4c 7b a4 a5 02 c9 02   03
8548 : d0 0d ad 3b 03 ac 3c 03   fe
8550 : 85 2b 84 2c 8a f0 e7 8a   f6
8558 : 30 dd 0a aa bd 26 a3 85   f5
8560 : 22 bd 27 a3 85 23 d0 0c   6c
8568 : 0a aa bd e5 81 85 22 bd   3c
8570 : e6 81 85 23 a9 0e 20 c3   f0
8578 : ff 20 cc ff a9 00 85 13   91
8580 : 20 d7 aa 20 45 ab a0 00   6f
8588 : b1 22 08 29 7f 20 d2 ff   b6
8590 : c8 28 10 f4 20 7a a6 a9   d3
8598 : 6a a0 a3 20 1e ab a4 3a   85
85a0 : c8 f0 94 20 c2 bd 20 d7   54
85a8 : aa ad fe 02 ae ff 02 85   27
85b0 : 5f 86 60 20 44 9a 20 d7   b8
85b8 : aa a9 91 20 d2 ff 4c 7b   f4
85c0 : a4 a9 00 85 0d 20 73 00   89
85c8 : b0 06 20 f3 bc 4c 5d 86   b3
85d0 : c9 2a d0 0d 20 73 00 a4   6b
85d8 : fb a5 fc 20 a8 88 4c 5d   a4
85e0 : 86 20 13 b1 90 06 20 b2   90
85e8 : 88 4c 5d 86 20 79 00 10   ac
85f0 : 03 4c b1 ae c9 25 f0 2b   3b
85f8 : c9 24 f0 24 c9 2d f0 1a   92
8600 : c9 22 f0 19 c9 21 f0 3f   21
8608 : c9 2b f0 b9 c9 3e f0 7e   29
8610 : c9 3c f0 7a 20 f1 ae 4c   68
8618 : 5d 86 4c 0d af 4c bd ae   1f
8620 : 4c 70 88 a9 00 85 3c 85   24
8628 : 3d 20 73 00 aa d0 07 a4   e9
8630 : 3c a5 3d 4c db 85 b0 f7   b4
8638 : 29 fe c9 30 d0 1a 8a 4a   f5
8640 : 26 3c 26 3d 4c 29 86 20   1e
8648 : 73 00 c9 4e d0 0a 20 73   bc
8650 : 00 c9 21 d0 03 4c d0 ae   ca
8658 : a9 00 4c 68 85 20 79 00   61
8660 : aa 10 01 60 68 68 8a c9   e6
8668 : 21 d0 0c 20 73 00 aa 20   1b
8670 : 73 00 c9 21 d0 e2 8a a2   0d
8678 : 0a dd 0f 81 f0 05 ca d0   69
```

Listing 1 »GIGA-ASS«, ein Assembler, der keine Wünsche offenläßt. Bitte mit dem MSE eingeben (siehe Seite 159).

```
8680 : f8 f0 04 8a 18 69 a9 a2  fb
8688 : 00 86 4d 4c bb ad 48 20  33
8690 : 73 00 20 f1 ae 20 f7 b7  85
8698 : aa 68 c9 3c f0 02 8a a8  0b
86a0 : 20 a2 b3 4c 5d 86 a9 00  38
86a8 : 85 3e 20 73 00 c9 23 f0  80
86b0 : 6f c9 28 f0 20 c9 26 d0  b7
86b8 : 71 20 4c 92 a5 3d d0 40  a7
86c0 : 20 fd ae c9 58 f0 08 a9  44
86c8 : 59 20 ff ae 4c 73 87 20  c6
86d0 : 73 00 4c 10 87 20 4c 92  28
86d8 : a5 3d d0 14 20 79 00 c9  34
86e0 : 2c f0 22 c9 29 d0 19 20  04
86e8 : 73 00 c9 2c f0 27 d0 03  e5
86f0 : 20 73 00 a5 41 c9 15 d0  d7
86f8 : 07 a9 09 85 3e 4c 73 87  ea
8700 : a9 04 4c 68 85 20 73 00  f3
8708 : a9 58 20 ff ae 20 f7 ae  0e
8710 : e6 3e 4c 73 87 20 73 00  de
8718 : a9 59 20 ff ae 4c 73 87  a0
8720 : a9 06 85 3e 20 5a 92 4c  ad
8728 : 73 87 a9 07 85 3e 20 79  68
8730 : 00 f0 3b c9 3b f0 37 a2  0e
8738 : 05 86 3e 20 4f 92 20 79  11
8740 : 00 c9 2c d0 22 20 73 00  3b
8748 : aa 20 73 00 8a c9 58 f0  19
8750 : 0b c9 59 f0 05 a9 02 4c  f2
8758 : 68 85 c6 3e a5 3d d0 13  aa
8760 : c6 3e c6 3e 4c 73 87 a9  91
8768 : 0a 85 3e 4c 5c 87 a9 07  05
8770 : 85 3e 60 20 79 00 f0 fa  82
8778 : c9 3b f0 f6 a9 03 4c 68  ae
8780 : 85 a6 3e e0 06 d0 0c a5  66
8788 : 41 c9 04 b0 06 a5 3b e9  13
8790 : 07 85 3b e0 09 f0 29 a5  4d
8798 : 41 c9 15 90 35 f0 14 c9  d4
87a0 : 16 f0 10 c9 30 90 03 4c  98
87a8 : 2e 88 e0 07 d0 1f a9 01  e2
87b0 : 85 42 60 e0 08 f0 04 e0  64
87b8 : 0a d0 12 a9 03 85 42 60  0a
87c0 : a5 41 c9 15 d0 07 a9 6c  e0
87c8 : 85 3b 4c bb 87 a9 04 4c  e4
87d0 : 68 85 a9 02 85 42 e0 08  a3
87d8 : f0 0d e0 0a d0 0a e6 42  46
87e0 : a5 3b 18 69 08 85 3b 60  b0
87e8 : a9 01 e0 00 f0 05 0a ca  3f
87f0 : 4c ea 87 a6 41 3d eb 80  17
87f8 : d0 11 a6 3e e0 02 f0 04  ac
8800 : e0 03 d0 c9 e8 e8 86 3e  3b
8808 : 4c e8 87 a6 3e e0 04 d0  1c
8810 : 0f a5 41 c9 02 d0 09 a9  99
8818 : 03 85 42 a9 be 85 3b 60  69
8820 : bd 08 81 85 42 bd 00 81  07
8828 : 18 65 3b 85 3b 60 20 b3  11
8830 : 87 c6 42 a5 fd c9 02 90  b7
8838 : f4 a5 3c a6 3d 38 e5 fb  08
8840 : a8 8a e5 fc 90 17 aa 98  e4
8848 : e9 02 b0 01 ca 85 3c e0  0a
8850 : 00 d0 05 c9 80 b0 01 60  85
8858 : a9 05 4c 68 85 aa 98 38  24
8860 : e9 02 b0 01 ca 85 3c e0  22
8868 : ff d0 ed c9 80 90 e9 60  79
8870 : a2 0a a9 00 95 5d ca d0  93
8878 : fb 20 73 00 90 0b 20 13  68
8880 : b1 90 17 e9 07 c9 40 b0  9d
8888 : 11 e9 2f 20 7e bd a5 61  8d
8890 : 18 69 04 85 61 90 e2 4c  cd
8898 : 7e b9 a5 61 e9 03 90 05  8b
88a0 : 85 61 4c 5d 86 4c 08 af  df
88a8 : 85 62 84 63 a2 90 38 4c  14
88b0 : 49 bc a5 7a a6 7b 85 49  ff
88b8 : 86 4a a2 00 20 79 00 20  1a
88c0 : 13 b1 b0 09 a9 20 85 81  ae
88c8 : a9 07 4c 68 85 20 73 00  3c
88d0 : 90 05 20 13 b1 90 04 e8  cf
88d8 : 4c cd 88 c9 5f f0 f8 e8  99
88e0 : 86 45 a9 36 85 01 a9 f9  35
88e8 : a2 bf 86 60 85 5f e4 30  5f
88f0 : 90 63 d0 04 c5 2f 90 5d  b9
88f8 : a0 01 b1 5f c9 ff f0 27  20
8900 : c9 fe d0 17 68 a8 68 48  5d
8908 : c9 90 f0 05 98 48 4c 48  84
8910 : 89 98 48 c9 8b d0 f7 a0  91
8918 : 02 d0 0d c5 8c d0 29 88  83
8920 : b1 5f c5 8b d0 22 c8 c8  36
8928 : b1 5f c5 45 d0 1a c8 b1  07
8930 : 5f 85 47 c8 b1 5f 85 48  f9
8938 : a0 00 b1 49 d1 47 d0 08  18
8940 : c8 c4 45 d0 f5 4c e6 89  46
8948 : a5 5f 38 e9 07 a6 60 b0  71
8950 : 9b ca 4c ea 88 68 aa 68  08
8958 : c9 85 d0 09 48 8a c9 e8  0b
8960 : d0 13 48 f0 1d c9 90 f0  2e
8968 : 06 48 8a 48 4c a4 89 48  df
8970 : 8a c9 8b f0 04 48 4c a4  dd
8978 : 89 48 20 e0 89 a9 0d 4c  fc
8980 : 68 85 20 e0 89 a5 fd c9  20
8988 : 02 90 05 a9 06 4c 68 85  b8
8990 : a5 3e c9 06 d0 07 a9 00  73
8998 : a0 80 4c a8 88 a9 80 a0  ba
89a0 : 00 4c a8 88 a5 5f a4 60  aa
89a8 : c0 a0 90 6d 85 2f 84 30  ce
89b0 : a0 00 a5 8b 91 5f c8 a5  ae
89b8 : 8c 91 5f c8 a5 45 91 5f  87
89c0 : c8 a5 49 91 5f c8 [illegible] 4a  47
89c8 : 91 5f c8 a9 ff 91 5f c8  0c
89d0 : 91 5f a5 5f a4 60 18 69  e7
89d8 : 05 90 01 c8 85 49 84 4a  c8
89e0 : a9 37 85 01 38 60 20 d2  53
89e8 : 89 a9 36 85 01 a0 01 b1  01
89f0 : 49 85 62 88 b1 49 85 63  e8
89f8 : c9 ff d0 13 a5 62 c9 ff  ec
8a00 : d0 0d a9 20 85 81 a9 37  3f
8a08 : 85 01 a9 06 4c 68 85 84  60
8a10 : 70 20 e0 89 20 ac 88 18  b3
8a18 : 60 20 e0 89 a9 20 85 81  a6
8a20 : a9 08 4c 68 85 a0 02 b1  b6
8a28 : 7a f0 1f c8 b1 7a 85 39  73
8a30 : c8 b1 7a 85 3a a4 7b a6  24
8a38 : 7a e8 d0 01 c8 8e fe 02  7c
8a40 : 8c ff 02 a0 04 20 fb a8  e3
8a48 : 18 60 38 60 a5 fd c9 02  20
8a50 : b0 03 4c b4 8b 20 cc ff  18
8a58 : a9 ff 85 3a a9 3f a0 83  c8
8a60 : 20 1e ab ad f9 02 f0 03  a9
8a68 : 20 42 94 ad f8 02 d0 06  73
8a70 : ae f7 02 20 c9 ff 20 e0  7d
8a78 : 92 a9 79 a0 83 20 1e ab  5a
8a80 : ad ec 02 18 6d ee 02 aa  d3
8a88 : ad ed 02 6d ef 02 20 79  dd
8a90 : 93 8a 20 79 93 a9 88 a0  89
8a98 : 83 20 1e ab 20 1f 93 20  b2
8aa0 : d7 aa a9 fa a0 83 20 1e  79
8aa8 : ab a5 2d 38 e5 2b 85 63  0d
8ab0 : a5 2e e5 2c 85 62 20 23  9d
8ab8 : 9a a9 34 a0 84 20 1e ab  61
8ac0 : a9 20 20 d2 ff a9 28 20  0a
8ac8 : d2 ff a5 2d 38 e5 2b a5  53
8ad0 : 2e e5 2c 4a 4a aa a9 00  e6
8ad8 : 20 cd bd a9 65 a0 84 20  31
8ae0 : 1e ab a9 0e a0 84 20 1e  eb
8ae8 : ab a5 fb 38 ed ec 02 85  c5
8af0 : 63 a5 fc ed ed 02 85 62  ec
8af8 : 20 23 9a a9 34 a0 84 20  20
8b00 : 1e ab a9 20 20 d2 ff a9  4e
8b08 : 28 20 d2 ff a5 fb 38 ed  ec
8b10 : ec 02 a5 fc ed ed 02 4a  f1
8b18 : 4a aa a9 00 20 cd bd a9  dc
8b20 : 65 a0 84 20 1e ab a9 21  23
8b28 : a0 84 20 1e ab a9 00 38  4f
8b30 : e5 2f 85 63 a9 c0 85 8f  50
8b38 : e5 30 85 62 20 23 9a a9  bc
8b40 : 34 a0 84 20 1e ab a9 20  10
8b48 : 20 d2 ff a9 28 20 d2 ff  d5
8b50 : a5 8e 38 e5 2f a5 8f e5  31
8b58 : 30 4a 4a aa a9 00 20 cd  4c
8b60 : bd a9 65 a0 84 20 1e ab  78
8b68 : ad f7 02 20 c3 ff 20 cc  ec
8b70 : ff ad eb 02 c9 ff f0 39  54
8b78 : a9 6a a0 84 20 1e ab a0  f2
8b80 : 00 84 cc a5 cb c9 40 f0  98
8b88 : fa a2 01 86 cc c9 3f d0  9e
8b90 : 08 a9 20 a4 d3 91 d1 d0  bc
8b98 : 18 c9 3c d0 e6 84 c6 a9  bf
8ba0 : 93 20 d2 ff ad ea 02 ae  8f
8ba8 : eb 02 85 14 86 15 20 30  6a
8bb0 : e1 4c 7b e3 ad ed 02 c9  f8
8bb8 : ff d0 05 a9 10 4c 68 85  a6
8bc0 : 20 ca 92 4c c0 8c a0 00  66
8bc8 : 84 8b 84 8c 84 8e c8 84  ad
8bd0 : 8d 60 24 9d 10 28 a9 00  33
8bd8 : 85 3e 20 73 00 d0 03 4c  1e
8be0 : bf 94 e9 80 b0 03 4c 08  d8
8be8 : af c9 0a f0 11 c9 13 90  e9
8bf0 : f5 c9 1b d0 03 4c 74 98  40
8bf8 : 20 f3 a7 4c ae a7 20 e7  fe
8c00 : ff a9 05 a2 08 86 7b 85  17
8c08 : 7a 20 79 00 c9 9e d0 09  d8
8c10 : 20 73 00 20 2a e1 4c 7b  c8
8c18 : e3 a9 2c a0 83 20 1e ab  f8
```



```
8c20 : a9 2f 85 00 ad 0e dc 09   93
8c28 : 80 8d 0e dc a9 08 8d 0f   bd
8c30 : dc 20 8e a6 a9 ff 8d ed   41
8c38 : 02 8d eb 02 a9 00 8d ee   eb
8c40 : 02 8d ef 02 a9 00 a2 c0   ec
8c48 : 86 30 85 2f 48 8d 0b dc   04
8c50 : 8d 0a dc 8d 09 dc 8d 08   89
8c58 : dc 85 fd 8d e8 02 85 fe   db
8c60 : 8d f9 02 8d fa 02 8d fb   0a
8c68 : 02 8d e7 02 20 c6 8b 8c   ea
8c70 : f8 02 20 25 8a 90 0d ad   d3
8c78 : fa 02 cd fb 02 f0 41 a9   66
8c80 : 0e 4c 68 85 20 73 00 c9   b0
8c88 : a1 f0 14 c9 a2 d0 28 ee   0f
8c90 : fb 02 ad fa 02 cd fb 02   d9
8c98 : b0 1d a9 09 4c 68 85 20   c1
8ca0 : 73 00 20 b2 88 a0 00 a5   4b
8ca8 : 7a 91 49 c8 a5 7b 91 49   65
8cb0 : a9 fe 91 5f ee fa 02 20   38
8cb8 : 09 a9 20 fb a8 4c 72 8c   ed
8cc0 : e6 fd a9 5f a0 83 20 1e   de
8cc8 : ab a5 fd 09 30 20 d2 ff   36
8cd0 : 20 84 94 20 e7 92 20 c6   7d
8cd8 : 8b 20 8e a6 20 cd 93 ad   06
8ce0 : e7 02 f0 17 ad fa 02 cd   3e
8ce8 : fb 02 f0 07 ad f8 02 09   be
8cf0 : 02 d0 05 ad f8 02 29 01   98
8cf8 : 8d f8 02 20 2c a8 20 25   59
8d00 : 8a 90 06 4c 4c 8a 4c c5   b3
8d08 : 8d ad f8 02 d0 32 a5 7a   14
8d10 : 48 a5 7b 48 20 73 00 aa   06
8d18 : 68 85 7b 68 85 7a e0 a2   24
8d20 : f0 1e ad fe 02 ae ff 02   04
8d28 : 85 5f 86 60 ae f7 02 20   fd
8d30 : c9 ff 90 03 4c f9 e0 20   d6
8d38 : 18 9a 20 7c 94 20 cc ff   b2
8d40 : a0 01 b1 7a 10 09 e6 7a   f6
8d48 : d0 02 e6 7b 4c dc 8d c9   b8
8d50 : 3b f0 b3 c9 20 f0 42 a5   07
8d58 : fd c9 01 f0 0f c8 b1 7a   8b
8d60 : f0 4f c9 20 d0 f7 20 fb   b3
8d68 : a8 4c 9c 8d a9 ff 85 81   c3
8d70 : 20 73 00 20 b2 88 a9 20   a4
8d78 : 85 81 b0 0d a5 4a c9 c0   e1
8d80 : 90 02 d0 05 a9 0a 4c 68   d3
8d88 : 85 a5 fc a6 fb 20 d5 92   31
8d90 : 20 79 00 aa d0 09 4c bc   c2
8d98 : 8d 20 73 00 20 79 00 20   20
8da0 : 81 92 a0 00 b1 7a f0 14   6d
8da8 : c9 3b f0 10 a9 03 4c 68   02
8db0 : 85 ad f8 02 d0 03 20 89   43
8db8 : 94 20 73 00 20 09 a9 20   6a
8dc0 : fb a8 4c dc 8c ad f8 02   dc
8dc8 : d0 ef 20 89 94 20 d3 93   8a
8dd0 : 20 73 00 20 09 a9 20 fb   04
8dd8 : a8 4c df 8c aa 29 e0 c9   3b
8de0 : a0 d0 07 8a 29 1f c9 19   e0
8de8 : 90 05 a9 01 4c 68 85 0a   b8
8df0 : aa bd 7b 82 48 bd 7a 82   09
8df8 : 48 e0 12 b0 04 e0 04 b0   04
8e00 : 08 ad f8 02 d0 03 4c 89   c7
8e08 : 94 60 20 73 00 20 b2 88   20
8e10 : a9 ff a0 01 91 5f 4c 1f   85
8e18 : 8e 20 73 00 20 b2 88 a9   a0
8e20 : 3d 20 ff ae a5 4a 48 a5   5c
8e28 : 49 48 20 4f 92 68 85 49   9c
8e30 : 68 85 4a a5 3d a6 3c 20   dc
8e38 : d5 92 ad f8 02 d0 28 ae   85
8e40 : f7 02 20 c9 ff 90 03 4c   a3
8e48 : f9 e0 a2 07 20 96 94 ae   a1
8e50 : f7 02 20 c9 ff 90 03 4c   b3
8e58 : f9 e0 a9 23 20 d2 ff a9   7c
8e60 : 3d 20 d2 ff 20 ab 93 4c   a8
8e68 : a2 8d 20 4c 92 8d ed 02   b4
8e70 : 8c ec 02 20 ca 92 4c a2   af
8e78 : 8d ad ed 02 c9 ff d0 05   81
8e80 : a9 10 4c 68 85 20 4c 92   01
8e88 : aa 98 38 ed ec 02 8d ee   3d
8e90 : 02 8a ed ed 02 8d ef 02   61
8e98 : 4c a2 8d a9 01 85 42 20   53
8ea0 : 5a 92 85 3b 20 6e 8f 20   00
8ea8 : e5 91 ad f8 02 d0 08 20   e7
8eb0 : d3 93 a9 80 8d f8 02 20   b0
8eb8 : 79 00 c9 2c f0 e1 4c 54   21
8ec0 : 8f a0 00 20 73 00 c9 22   46
8ec8 : f0 03 4c 08 af a2 00 c8   f0
8ed0 : b1 7a f0 33 c9 5f d0 02   40
8ed8 : a9 0d c9 22 f0 23 95 3b   b3
8ee0 : e8 e0 03 d0 ea 86 42 98   30
8ee8 : 48 20 6e 8f 20 e5 91 ad   a1
8ef0 : f8 02 d0 08 20 d3 93 a9   61
8ef8 : 80 8d f8 02 68 a8 4c cd   56
8f00 : 8e c8 a9 00 95 3b e8 8a   49
8f08 : d0 06 c0 01 f0 bc d0 19   96
8f10 : 86 42 98 48 20 6e 8f 20   da
8f18 : e5 91 ad f8 02 d0 08 20   57
8f20 : d3 93 a9 80 8d f8 02 68   b1
8f28 : a8 20 fb a8 4c 54 8f a9   ed
8f30 : 02 85 42 20 4c 92 85 3c   71
8f38 : 84 3b 20 6e 8f 20 e5 91   e4
8f40 : ad f8 02 d0 08 20 d3 93   fc
8f48 : a9 80 8d f8 02 20 79 00   bb
8f50 : c9 2c f0 df ad f8 02 29   64
8f58 : 01 8d f8 02 a0 00 b1 7a   64
8f60 : f0 09 c9 3b f0 05 a9 03   92
8f68 : 4c 68 85 4c d3 8d a6 fd   13
8f70 : ca f0 2b ad f8 02 d0 03   1c
8f78 : 4c 90 93 10 21 ae f7 02   5f
8f80 : 20 c9 ff 90 03 4c f9 e0   d3
8f88 : a2 08 20 96 94 20 90 93   bd
8f90 : ae f7 02 20 c9 ff 90 03   a3
8f98 : 4c f9 e0 4c eb 93 60 20   c0
8fa0 : 4c 92 c9 00 f0 03 4c 48   91
8fa8 : b2 85 42 c0 00 f0 a5 85   ef
8fb0 : 3b 88 e6 42 c0 00 f0 0e   1d
8fb8 : 85 3c 88 e6 42 c0 00 f0   66
8fc0 : 05 85 3d 88 e6 42 98 48   5c
8fc8 : 20 6e 8f 20 e5 91 ad f8   9b
8fd0 : 02 d0 08 20 d3 93 a9 80   c2
8fd8 : 8d f8 02 68 a8 a9 00 f0   29
8fe0 : c8 a6 fd ca d0 07 a9 80   c1
8fe8 : 85 fe 4c bc 8d 20 73 00   3f
8ff0 : a6 7a a4 7b e8 d0 01 c8   16
8ff8 : 86 bb 84 bc a0 00 c8 b1   a5
9000 : 7a c9 22 d0 f9 88 84 b7   67
9008 : a9 0e a0 6e a2 08 20 ba   0f
9010 : ff 20 c0 ff 90 03 4c f9   96
9018 : e0 a2 0e 20 c9 ff a5 fb   fc
9020 : 20 d2 ff a5 fc 20 d2 ff   7a
9028 : 20 cc ff a5 90 d0 03 4c   97
9030 : bc 8d 20 e7 ff a9 de a0   c2
9038 : 83 20 1e ab 4c ca 9c a5   a1
9040 : fd c9 02 90 0c a9 00 85   ce
9048 : fe a9 0e 20 c3 ff 20 cc   f9
9050 : ff 4c bc 8d 20 09 a9 20   87
9058 : fb a8 a0 04 20 fb a8 a5   20
9060 : 7b c5 2e 90 0b a5 7a c5   af
9068 : 2d 90 05 a9 0e 4c 68 85   44
9070 : 20 73 00 c9 a2 d0 dd 4c   44
9078 : bc 8d 20 73 00 ee fa 02   d9
9080 : a5 8b 05 8c 85 8f 20 79   06
9088 : 00 20 b2 88 a5 7b 48 8d   c8
9090 : fd 02 a5 7a 48 8d fc 02   30
9098 : a9 36 85 01 48 a0 01 b1   cf
90a0 : 49 85 7b 88 b1 49 85 7a   0c
90a8 : a9 37 85 01 20 79 00 c9   d0
90b0 : a0 d0 66 20 d0 91 20 79   63
90b8 : 00 c9 a0 d0 62 20 73 00   d4
90c0 : c9 22 d0 09 20 6a 92 a9   e3
90c8 : 00 85 3d f0 03 20 4f 92   8c
90d0 : 20 d0 91 20 24 91 20 73   f7
90d8 : 00 20 b2 88 a5 3d a6 3c   fd
90e0 : 20 d5 92 20 79 00 c9 00   52
90e8 : f0 18 c9 3b f0 14 c9 2c   ed
90f0 : d0 2d 20 d0 91 20 79 00   79
90f8 : c9 2c d0 23 20 3f 91 4c   4b
9100 : bd 90 20 d0 91 20 79 00   27
9108 : c9 00 f0 04 c9 3b d0 0f   66
9110 : 20 d0 91 20 38 91 4c bc   bb
9118 : 8d 20 24 91 4c 13 91 a9   e8
9120 : 0b 4c 68 85 a5 8f d0 09   48
9128 : a5 8d a6 8e 85 8b 86 8c   f7
9130 : 60 e6 8b d0 02 e6 8c 60   4b
9138 : e6 8d d0 02 e6 8e 60 a5   09
9140 : 8f d0 07 a9 00 85 8b 85   94
9148 : 8c 60 a5 8b d0 02 c6 8c   30
9150 : c6 8b 60 ee fb 02 68 c9   d7
9158 : 36 d0 1b 68 85 7a 68 85   a3
9160 : 7b 68 48 c9 36 d0 06 20   9d
9168 : 4a 91 4c d3 8d a9 00 85   3a
9170 : 8b 85 8c 4c d3 8d a9 0c   d3
9178 : 4c 68 85 20 73 00 20 8a   2b
9180 : ad a5 61 f0 0c 20 fd ae   8d
9188 : 20 4f 92 20 a3 a8 4c dc   63
9190 : 8c 4c bc 8d 20 4c 92 4c   6a
9198 : 8b 91 20 73 00 20 8a ad   e9
91a0 : a5 61 f0 03 4c a2 8d 20   e3
91a8 : 09 a9 c8 c8 c8 c8 20 fb   1c
91b0 : a8 a5 7b c5 2e 90 0b a5   a1
91b8 : 7a c5 2d 90 05 a9 0f 4c   e5
91c0 : 68 85 20 73 00 c9 b0 f0   54
91c8 : 04 c9 af d0 da 4c bc 8d   d5
91d0 : a6 7a a4 7b ad fc 02 85   22
91d8 : 7a ad fd 02 85 7b 8e fc   51
91e0 : 02 8c fd 02 60 a6 fd ca   b1
91e8 : f0 2d a5 fe 30 35 d0 27   f6
91f0 : a5 fb 18 6d ee 02 85 22   a0
91f8 : a5 fc 6d ef 02 85 23 a6   9b
9200 : 42 a0 00 a5 3b 91 22 ca   a5
9208 : f0 0d c8 a5 3c 91 22 ca   d4
9210 : f0 05 c8 a5 3d 91 22 a5   9e
9218 : 42 18 65 fb 85 fb 90 02   be
9220 : e6 fc 60 a2 0e 20 c9 ff   fa
9228 : a6 42 a5 3b 20 d2 ff ca   ee
9230 : f0 0d a5 3c 20 d2 ff ca   c6
9238 : f0 05 a5 3d 4c 34 92 20   ad
9240 : cc ff a5 90 f0 03 4c 32   44
9248 : 90 4c 17 92 20 73 00 20   f4
9250 : 8a ad 20 f7 b7 85 3d 84   5d
9258 : 3c 60 20 73 00 c9 22 f0   f4
9260 : 09 20 4f 92 c9 00 d0 16   ab
9268 : 98 60 a0 01 b1 7a 85 3c   f6
9270 : c8 b1 7a c9 22 d0 07 c8   3f
9278 : 20 fb a8 a5 3c 60 4c 48   fd
9280 : b2 a0 00 b1 7a c9 c0 b0   13
9288 : 18 c9 80 b0 0d 68 68 ad   cc
9290 : f8 02 d0 03 20 89 94 4c   57
9298 : 7d 90 aa 68 68 8a 4c dc   db
92a0 : 8d c9 f8 90 05 a9 01 4c   9c
92a8 : 68 85 29 3f 85 41 aa bd   8e
92b0 : b3 80 85 3b 20 a6 86 20   fe
92b8 : 81 87 a6 fd ca f0 08 ad   16
92c0 : f8 02 d0 03 20 90 93 4c   bb
92c8 : e5 91 ad ec 02 ae ed 02   d0
92d0 : 85 fb 86 fc 60 a0 01 91   c6
92d8 : 49 88 8a 91 49 84 70 60   75
92e0 : a9 65 a0 83 20 1e ab 20   b6
92e8 : 84 94 ad 08 dc ad 0b dc   44
92f0 : ad 0a dc 20 07 93 ad 09   b3
92f8 : dc 20 07 93 ad 08 dc 09   b9
9300 : 30 20 d2 ff 4c d7 aa aa   78
9308 : 4a 4a 4a 4a 09 30 20 d2   8b
9310 : ff 8a 29 0f 09 30 20 d2   b9
9318 : ff a9 3a 20 d2 ff 60 a5   78
9320 : fb 18 6d ee 02 aa a5 fc   66
9328 : 6d ef 02 20 79 93 8a 4c   08
9330 : 79 93 a6 fd ca f0 2d 20   05
9338 : 73 00 20 19 e2 a5 b8 8d   30
9340 : f7 02 20 c0 ff b0 20 a9   b2
9348 : 00 8d f8 02 ae f7 02 20   80
9350 : c9 ff 90 03 4c f9 e0 a9   09
9358 : 8d a0 83 20 1e ab 20 d7   8a
9360 : aa 20 cc ff 4c d3 8d 4c   80
9368 : f9 e0 20 4c 92 98 09 80   76
9370 : 8d e8 02 8d e9 02 4c a2   c9
9378 : 8d 48 4a 4a 4a 4a 20 84   86
9380 : 93 68 29 0f 18 69 f6 90   3d
9388 : 02 69 06 69 3a 4c d2 ff   3f
9390 : ae f7 02 20 c9 ff 90 03   a3
9398 : 4c f9 e0 20 1f 93 20 7c   25
93a0 : 94 a6 42 a5 3b 20 79 93   8e
93a8 : ca f0 16 20 84 94 a5 3c   70
93b0 : 20 79 93 ca f0 0e 20 84   d4
93b8 : 94 a5 3d 20 79 93 4c c7   67
93c0 : 93 20 7c 94 20 7c 94 20   8d
93c8 : 7c 94 4c cc ff ad f8 02   90
93d0 : f0 01 60 ad fe 02 ae ff   c9
93d8 : 02 85 5f 86 60 ae f7 02   a5
93e0 : 20 c9 ff 90 03 4c f9 e0   33
93e8 : 20 47 9a 2c e8 02 10 2c   0f
93f0 : ce e9 02 30 27 ad e8 02   c1
93f8 : 8d e9 02 a5 ba c9 03 d0   57
9400 : 0f 20 1f 9b a9 00 85 d4   b5
9408 : a9 93 20 d2 ff 4c 15 94   bd
9410 : a9 0c 20 d2 ff a9 8d a0   e6
9418 : 83 20 1e ab 20 d7 aa 4c   ac
9420 : cc ff a6 fd ca f0 15 20   1e
9428 : 73 00 20 19 e2 a2 05 b5   89
9430 : b7 9d f0 02 ca 10 f8 a9   97
9438 : ff 8d f9 02 4c bc 8d 4c   36
9440 : f9 e0 a2 05 bd f0 02 95   89
9448 : b7 ca 10 f8 20 c0 ff b0   f1
9450 : ee ae f1 02 20 c9 ff b0   04
9458 : e6 a9 ac a0 83 20 1e ab   5b
9460 : 20 1f 9d ad f1 02 20 c3   64
9468 : ff 4c cc ff a6 fd ca f0   28
9470 : cb 20 cd 93 a9 0e 20 c3   44
9478 : ff 4c 4c 8a a9 20 20 d2   c3
9480 : ff 20 d2 ff a9 20 4c d2   b6
9488 : ff ae f7 02 20 c9 ff 90   8e
9490 : 03 4c f9 e0 a2 12 20 84   98
9498 : 94 ca d0 fa 4c cc ff a9   a3
94a0 : 01 85 fe 4c bc 8d 20 4c   fe
94a8 : 92 8d eb 02 8c ea 02 4c   fd
94b0 : a2 8d a9 01 8d e7 02 4c   5c
94b8 : bc 8d a9 00 85 02 60 a9   e2
94c0 : ff 85 3a a6 02 f0 0c 30   1d
94c8 : 75 ca ca f0 68 ca f0 68   e4
94d0 : ca f0 68 20 60 a5 86 7a   73
94d8 : 84 7b 20 73 00 aa f0 f3   91
94e0 : c9 2f d0 09 20 73 00 20   74
94e8 : a0 aa 4c 37 85 c9 5f d0   9d
94f0 : 24 20 73 00 20 79 00 b0   30
```

Listing 1. »Giga Ass« (Fortsetzung)

```
94f8 : 03 4c e2 98 a2 15 dd ac  91
9500 : 82 f0 06 ca 10 f8 4c 08  df
9508 : af 8a 0a aa bd c2 82 48  61
9510 : bd c1 82 48 60 20 79 00  44
9518 : b0 03 4c c0 95 c9 40 f0  ff
9520 : d3 c9 3f f0 0a a0 01 b1  d3
9528 : 7a f0 c9 c9 41 90 c5 20  b6
9530 : 79 a5 4c e1 a7 4c 4a 85  dc
9538 : 4c e8 96 4c 8e 9b a5 02  88
9540 : 29 7f 85 02 a9 91 20 d2  18
9548 : ff a2 00 20 84 94 e8 e0  ef
9550 : 27 90 f8 a9 00 85 d3 20  ee
9558 : 44 9a c6 d6 a5 d5 48 20  e0
9560 : d7 aa 68 c9 27 d0 03 20  25
9568 : d7 aa 4c bf 94 98 48 a0  10
9570 : 00 84 41 8a 48 e8 bd 00  17
9578 : 02 10 0f 29 7f 85 3b b9  f0
9580 : 07 82 29 7f c5 3b d0 14  a4
9588 : f0 25 b9 07 82 08 29 7f  66
9590 : dd 00 02 d0 06 28 30 17  99
9598 : c8 d0 da 28 b9 07 82 30  c2
95a0 : 03 c8 d0 f8 68 aa c8 c0  db
95a8 : 73 b0 0e e6 41 d0 c4 68  52
95b0 : 68 a8 a5 41 09 a0 4c 9f  04
95b8 : 96 68 a8 a9 2e 4c 9f 96  d3
95c0 : a2 ff 86 81 20 6b a9 a9  8b
95c8 : 20 85 81 a6 7a a0 05 bd  1c
95d0 : 00 02 99 fb 01 10 0d 29  ce
95d8 : bf 09 20 4c 9f 96 bd 00  53
95e0 : 02 99 fb 01 d0 03 4c d5  d0
95e8 : 96 c9 3b d0 03 4c b8 96  ee
95f0 : c9 2e f0 28 c9 20 f0 04  7b
95f8 : e8 c8 d0 e2 e8 bd 00 02  55
9600 : d0 03 4c d1 96 c9 20 f0  b9
9608 : f3 c9 22 d0 03 4c c0 96  45
9610 : c9 3b d0 03 4c b4 96 c9  64
9618 : 2e d0 04 c8 4c 6d 95 98  80
9620 : 48 a0 00 84 41 bd 00 02  4f
9628 : d9 0b 80 d0 10 bd 01 02  b8
9630 : d9 0c 80 d0 08 bd 02 02  c4
9638 : d9 0d 80 f0 59 e6 41 c8  39
9640 : c8 c8 c0 a8 d0 df 68 a8  b0
9648 : bd 00 02 c9 a1 b0 1d c9  67
9650 : a0 d0 3c e8 bd 00 02 ca  fe
9658 : 20 13 b1 a9 a0 90 3d b0  88
9660 : 2e bd 00 02 f0 6b 10 0c  70
9668 : c9 a0 f0 32 c9 ff d0 2c  3c
9670 : a9 de 30 2a c9 3b f0 3c  8c
9678 : c9 20 d0 13 e8 bd 00 02  68
9680 : f0 4f c9 20 f0 f6 c9 3b  f3
9688 : f0 2a ca a9 a0 30 0f c8  ce
9690 : 99 fb 01 e8 d0 cb 68 a8  e3
9698 : e8 e8 a5 41 09 c0 c8 99  73
96a0 : fb 01 e8 bd 00 02 f0 29  34
96a8 : c9 20 f0 f6 c9 22 f0 10  2e
96b0 : c9 3b d0 ea c8 99 fb 01  f3
96b8 : e8 bd 00 02 d0 f6 f0 11  6a
96c0 : c8 99 fb 01 e8 bd 00 02  f4
96c8 : f0 07 c9 22 d0 f2 4c 9e  05
96d0 : 96 c8 99 fb 01 c6 7b a9  38
96d8 : ff 85 7a c0 06 90 06 a5  99
96e0 : 02 09 80 85 02 4c a2 a4  8e
96e8 : ad 3d 03 f0 5c ad fe 02  46
96f0 : cd 3b 03 f0 0d ad 3c 03  6f
96f8 : 85 62 ad 3b 03 85 63 20  ab
9700 : 23 9a a2 05 78 bd 76 02  0d
9708 : 9d 7b 02 ca d0 f7 86 d3  cb
9710 : a0 04 a9 1d 99 77 02 e6  eb
9718 : c6 88 10 f8 58 20 60 a5  99
9720 : 86 7a 84 7b 20 73 00 b0  73
9728 : 20 a2 ff 86 81 20 6b a9  84
9730 : ad 3b 03 8d fe 02 c5 14  2d
9738 : d0 0c 18 6d 3d 03 8d 3b  5b
9740 : 03 90 03 ee 3c 03 4c c7  c7
9748 : 95 20 ba 94 4c bf 94 e6  11
9750 : 7a a0 00 b1 7a f0 15 20  14
9758 : eb b7 a5 14 a4 15 8d 3b  aa
9760 : 03 8c 3c 03 8e 3d 03 aa  4d
9768 : e8 8e fe 02 a9 03 85 02  64
9770 : 4c bf 94 20 73 00 90 0d  59
9778 : f0 0b c9 2d f0 07 c9 ab  cc
9780 : f0 03 4c 08 af 20 6b a9  03
9788 : 20 13 a6 20 79 00 f0 10  5b
9790 : c9 2d f0 04 c9 ab d0 ea  bf
9798 : 20 73 00 20 6b a9 d0 e2  83
97a0 : a5 14 05 15 d0 06 a9 ff  17
97a8 : 85 14 85 15 60 a0 01 b1  ae
97b0 : 5f f0 13 c8 b1 5f aa c8  b8
97b8 : b1 5f c5 15 d0 04 e4 14  16
97c0 : f0 02 b0 03 18 24 38 60  82
97c8 : a0 00 b1 5f aa c8 b1 5f  37
97d0 : 85 60 86 5f 60 4c 48 b2  02
97d8 : a9 bc a0 83 20 1e ab a9  6d
97e0 : 00 85 cc a5 c6 f0 fc a9  c6
97e8 : 01 85 cc a9 00 85 c6 ad  b7
97f0 : 77 02 c9 59 f0 08 a9 4e  98
97f8 : 20 d2 ff 4c 37 85 a9 59  04
9800 : 20 d2 ff 20 73 97 a5 5f  d6
9808 : a6 60 85 19 86 1a a5 14  5b
9810 : 25 15 aa e8 f0 06 e6 14  8a
9818 : d0 02 e6 15 20 13 a6 a5  c6
9820 : 5f a6 60 85 24 86 25 38  17
9828 : e5 19 8a e5 1a 90 a6 a5  05
9830 : 2d e5 24 85 5f a5 2e e5  b1
9838 : 25 85 60 18 a5 19 65 5f  b2
9840 : 85 2d a5 1a 65 60 85 2e  d4
9848 : a0 00 a5 5f 38 e9 01 85  1f
9850 : 5f a5 60 e9 00 85 60 30  e5
9858 : 12 b1 24 91 19 e6 19 d0  4d
9860 : 02 e6 1a e6 24 d0 02 e6  d7
9868 : 25 d0 df 20 59 a6 20 33  a3
9870 : a5 4c 37 85 20 73 97 20  f6
9878 : 2c a8 20 ad 97 b0 0c 20  26
9880 : 44 9a 20 d7 aa 20 c8 97  12
9888 : 4c 77 98 4c 37 85 20 9b  97
9890 : b7 20 d9 98 86 fe 20 fd  be
9898 : ae 20 8a ad 20 f7 b7 a6  9d
98a0 : fe 9d 50 03 98 9d 40 03  5f
98a8 : 20 fd ae 20 8a ad 20 f7  fd
98b0 : b7 a6 fe 9d 70 03 98 9d  ea
98b8 : 60 03 4c 37 85 bd 40 03  e1
98c0 : 85 14 bd 50 03 85 15 86  86
98c8 : fe 20 13 a6 a6 fe bd 60  8a
98d0 : 03 85 14 bd 70 03 85 15  b2
98d8 : 60 e0 10 90 fb 68 68 4c  fc
98e0 : 48 b2 20 9e b7 20 d9 98  72
98e8 : 20 bd 98 a9 93 20 d2 ff  c8
98f0 : 4c 77 98 20 9b b7 20 d9  ce
98f8 : 98 86 fe 20 fd ae 20 eb  45
9900 : b7 8e 3d 03 a5 14 a6 15  6e
9908 : 8d 3b 03 8e 3c 03 a6 fe  3a
9910 : 20 bd 98 ad 3b 03 9d 40  ad
9918 : 03 ad 3c 03 9d 50 03 20  0a
9920 : ad 97 b0 1d ad 3c 03 91  54
9928 : 5f 88 ad 3b 03 91 5f 18  09
9930 : 6d 3d 03 8d 3b 03 90 03  c2
9938 : ee 3c 03 20 c8 97 4c 1f  c2
9940 : 99 ad 3b 03 38 ed 3d 03  cd
9948 : a6 fe 9d 60 03 b0 03 ce  40
9950 : 3c 03 ad 3c 03 9d 70 03  e6
9958 : 4c 37 85 a9 02 85 02 a5  76
9960 : 2b a6 2c 8d 3b 03 8e 3c  1a
9968 : 03 a9 ff 85 14 85 15 20  f2
9970 : 13 a6 a5 5f a6 60 [illegible] 2b  05
9978 : 86 2c 4c 93 99 20 73 00  02
9980 : f0 04 c9 22 d0 f7 20 57  25
9988 : e2 a0 00 a2 08 4c ba ff  dd
9990 : a9 01 2c a9 00 85 0a 20  8f
9998 : 7d 99 20 6f e1 4c 37 85  40
99a0 : 20 7d 99 20 59 e1 4c 37  2d
99a8 : 85 20 9b b7 20 d9 98 20  8f
99b0 : bd 98 20 fd ae 20 ee 9b  60
99b8 : 20 ad 97 90 03 4c 37 85  21
99c0 : a6 60 a5 5f 18 69 04 85  d4
99c8 : 5d 90 01 e8 86 5e a9 03  d3
99d0 : 85 45 a0 ff e6 45 c8 b1  3f
99d8 : 5d f0 24 c4 ba f0 24 d9  c6
99e0 : d8 03 f0 f2 b9 d8 03 c9  d6
99e8 : 3f f0 eb c8 b1 5d d0 fb  f4
99f0 : c4 ba f0 1e 90 1c e6 5d  51
99f8 : d0 02 e6 5e 4c d2 99 c4  9a
9a00 : ba d0 0f a5 02 c9 04 d0  bb
9a08 : 03 4c 49 9b 20 44 9a 20  c6
9a10 : d7 aa 20 c8 97 4c b8 99  4f
9a18 : a0 02 b1 5f 85 63 c8 b1  0c
9a20 : 5f 85 62 a2 90 38 20 49  0d
9a28 : bc 20 df bd 20 87 b4 20  f5
9a30 : a6 b6 85 b4 aa e0 05 b0  50
9a38 : 06 20 84 94 e8 d0 f6 a6  40
9a40 : b4 4c 25 ab 20 18 9a a2  4c
9a48 : 00 a0 04 b1 5f 30 42 c9  e4
9a50 : 3b d0 03 4c 09 9b b1 5f  31
9a58 : d0 03 4c 13 9b c9 20 f0  8a
9a60 : 07 20 d2 ff e8 c8 d0 ee  22
9a68 : 20 d2 ff c8 e8 ec 3e 03  ff
9a70 : b0 06 20 84 94 4c 6c 9a  4f
9a78 : b1 5f 30 0d f0 71 c9 3b  bf
9a80 : f0 7c 20 d2 ff e8 c8 d0  1d
9a88 : ef c9 f8 b0 62 c9 c0 b0  89
9a90 : 38 29 1f f0 5a c9 19 90  bc
9a98 : 04 a9 3f d0 5a 85 22 a9  08
9aa0 : 2e 20 d2 ff e8 98 48 a0  49
9aa8 : 00 c6 22 30 09 c8 b9 07  66
9ab0 : 82 10 fa c8 d0 f3 b9 07  b4
9ab8 : 82 08 29 7f 20 d2 ff e8  e3
9ac0 : c8 28 10 f2 68 a8 c8 d0  8f
9ac8 : 22 29 3f 85 22 0a 65 22  4c
9ad0 : 84 22 a8 b9 0b 80 20 d2  a1
9ad8 : ff e8 b9 0c 80 20 d2 ff  90
9ae0 : e8 b9 0d 80 20 d2 ff e8  63
9ae8 : a4 22 c8 20 84 94 e8 b1  c7
9af0 : 5f f0 20 c9 3b f0 07 20  a0
9af8 : d2 ff e8 c8 d0 f1 ec 3f  ec
9b00 : 03 b0 06 20 84 94 e8 d0  13
9b08 : f5 b1 5f f0 06 20 d2 ff  78
9b10 : c8 d0 f6 a5 cb c9 3c d0  50
9b18 : 19 a5 cb c9 40 d0 fa 20  e7
9b20 : 2c a8 a5 cb c9 3c d0 f7  35
9b28 : a5 cb c9 40 d0 fa a9 00  b9
9b30 : 85 c6 60 20 9b b7 20 d9  e0
9b38 : 98 20 bd 98 20 fd ae 20  50
9b40 : cc 9b a9 04 85 02 4c b2  c4
9b48 : 99 a5 14 48 a5 15 48 a2  2b
9b50 : 05 a0 02 b1 5f 85 14 c8  60
9b58 : b1 5f 85 15 c8 c4 45 f0  66
9b60 : 09 b1 5f 9d fb 01 e8 4c  d1
9b68 : 5c 9b a0 00 c4 b7 f0 0b  9e
9b70 : b9 b0 03 9d fb 01 e8 c8  f3
9b78 : 4c 6c 9b a4 ba b1 5d 9d  60
9b80 : fb 01 f0 05 e8 c8 4c 7d  da
9b88 : 9b 8a a8 4c a2 a4 20 13  12
9b90 : a6 20 44 9a 20 d7 aa 68  e7
9b98 : 85 15 68 85 14 a6 fe bd  61
9ba0 : 60 03 85 14 bd 70 03 85  dc
9ba8 : 15 a5 45 18 65 b7 a8 b1  fe
9bb0 : 5f f0 13 a6 60 98 18 65  17
9bb8 : 5f 85 5d 90 01 e8 86 5e  71
9bc0 : 88 84 45 4c d2 99 4c 12  b4
9bc8 : 9a 4c 48 b2 20 9e ad 20  df
9bd0 : 82 b7 c9 26 b0 f3 85 b7  95
9bd8 : a2 00 a0 00 20 12 9c 90  c9
9be0 : 04 c6 b7 c6 b7 9d b0 03  3f
9be8 : e8 e4 b7 90 ef 60 20 9e  02
9bf0 : ad 20 82 b7 f0 d3 c9 26  66
9bf8 : b0 cf 85 ba a2 00 a0 00  f5
9c00 : 20 12 9c 90 04 c6 ba c6  51
9c08 : ba 9d d8 03 e8 e4 ba 90  e9
9c10 : ef 60 8a 48 a2 00 86 41  a2
9c18 : b1 22 dd 0b 80 d0 12 c8  1b
9c20 : b1 22 dd 0c 80 d0 09 c8  1f
9c28 : b1 22 dd 0d 80 f0 12 88  ec
9c30 : 88 e8 e8 e8 e6 41 e0 a8  d1
9c38 : d0 de 68 aa b1 22 c8 18  66
9c40 : 60 68 aa a5 41 09 c0 c8  25
9c48 : 38 60 a9 24 8d 00 01 20  bc
9c50 : d7 aa a9 01 a8 a2 00 20  e7
9c58 : bd ff a2 08 a0 60 20 ba  c1
9c60 : ff 20 d5 f3 a5 ba 20 b4  7d
9c68 : ff a5 b9 20 96 ff a9 00  bc
9c70 : 85 90 a0 03 84 ff 20 a5  da
9c78 : ff 85 fe a4 90 d0 39 20  43
9c80 : a5 ff a4 90 d0 32 a4 ff  91
9c88 : 88 d0 e9 a6 fe 20 cd bd  6b
9c90 : 20 84 94 20 a5 ff a6 90  32
9c98 : d0 1e a8 aa f0 06 20 d2  5c
9ca0 : ff 4c 93 9c 20 d7 aa ad  04
9ca8 : 01 dc 29 10 f0 0a ad 01  7c
9cb0 : dc 10 05 a0 02 4c 74 9c  77
9cb8 : 20 42 f6 4c ca 9c a0 01  57
9cc0 : b1 7a f0 06 c9 24 f0 84  36
9cc8 : d0 22 a9 08 85 ba 20 b4  2d
9cd0 : ff a9 6f 85 b9 20 96 ff  27
9cd8 : 20 a5 ff 20 d2 ff c9 0d  3d
9ce0 : d0 f6 20 ab ff a9 00 85  01
9ce8 : c6 4c 37 85 a9 08 85 ba  b9
9cf0 : 20 b1 ff a9 6f 85 b9 20  68
9cf8 : 93 ff b1 7a f0 06 20 a8  58
9d00 : ff c8 d0 f6 20 ae ff 4c  86
9d08 : 37 85 a5 30 c5 2e d0 06  8e
9d10 : a5 2f c5 2d f0 03 20 1f  4a
9d18 : 9d 4c 37 85 4c d7 aa a9  db
9d20 : f9 a2 bf 86 46 85 45 e4  9a
9d28 : 30 90 f1 d0 04 c5 2f 90  83
9d30 : eb 20 2c a8 20 d7 aa a9  0a
9d38 : 59 a0 83 20 1e ab a9 36  19
9d40 : 85 01 a0 06 b1 45 85 3d  05
9d48 : 88 b1 45 85 3c 88 b1 45  04
9d50 : 85 23 88 b1 45 85 22 a9  1b
9d58 : 37 85 01 a5 2b a6 2c 85  ea
9d60 : 5f 86 60 a0 01 b1 5f aa  9f
9d68 : 88 c5 23 90 0a f0 02 b0  3f
9d70 : 0e b1 5f c5 22 b0 08 b1  13
9d78 : 5f 85 5f 86 60 90 e4 20  a1
9d80 : 18 9a a9 3a 20 d2 ff 20  70
9d88 : 84 94 a9 36 85 01 a0 04  72
9d90 : b1 45 85 23 88 b1 45 85  e0
9d98 : 22 88 b1 45 85 47 88 b1  2b
9da0 : 45 85 62 88 b1 45 85 63  74
9da8 : a9 37 85 01 a2 00 b1 22  a4
9db0 : 20 d2 ff c8 e8 e0 0e b0  81
9db8 : 0e c4 47 90 f1 e8 a9 2e  76
9dc0 : 20 d2 ff e0 0e 90 f6 a9  fa
9dc8 : 2e 20 d2 ff a9 24 20 d2  9d
9dd0 : ff a5 3d 20 79 93 a5 3c  38
9dd8 : 20 79 93 20 84 94 a6 62  ea
9de0 : 10 17 e8 d0 0a a9 ec a0  b3
9de8 : 83 20 1e ab 4c ff 9d a9  07
9df0 : f3 a0 83 20 1e ab 4c ff  89
```

```
9df8 : 9d 20 84 94 20 23 9a 20  1f
9e00 : 13 9b a5 45 a6 46 38 e9  44
9e08 : 07 b0 01 ca a0 37 84 01  d9
9e10 : 4c 23 9d a9 ff a0 01 91  b6
9e18 : 2b 20 33 a5 a5 22 a4 23  19
9e20 : 18 69 02 90 01 c8 85 2d  46
9e28 : 84 2e 20 60 a6 a9 3b a0  bd
9e30 : 84 20 1e ab ad 82 02 20  f9
9e38 : 79 93 ad 81 02 20 79 93  45
9e40 : a9 88 a0 83 20 1e ab a5  b3
9e48 : 38 20 79 93 a5 37 20 79  e9
9e50 : 93 a9 20 20 d2 ff a9 28  e8
9e58 : 20 d2 ff a5 37 38 ed 81  86
9e60 : 02 a5 38 ed 82 02 4a 4a  f7
9e68 : aa a9 00 20 cd bd a9 65  27
9e70 : a0 84 20 1e ab a9 50 a0  a9
9e78 : 84 20 1e ab a5 2c 20 79  39
9e80 : 93 a5 2b 20 79 93 a9 88  a1
9e88 : a0 83 20 1e ab a5 2e 20  97
9e90 : 79 93 a5 2d 20 79 93 a9  51
9e98 : 20 20 d2 ff a9 28 20 d2  7f
9ea0 : ff a5 2d 38 e5 2b a5 2e  6f
9ea8 : e5 2c 4a 4a aa a9 00 20  b7
9eb0 : cd bd a9 65 a0 84 20 1e  5e
9eb8 : ab 20 d7 aa a5 2d 38 e5  2f
9ec0 : 2b 85 63 a5 2e e5 2c 85  09
9ec8 : 62 20 23 9a a9 cc a0 83  e1
9ed0 : 20 1e ab a5 37 38 e5 2d  c6
9ed8 : 85 63 a5 38 e5 2e 85 62  2a
9ee0 : 20 23 9a a9 66 a0 e4 20  ad
9ee8 : 1e ab 4c 37 85 20 4c 92  86
9ef0 : a5 15 30 2d e6 14 d0 02  28
9ef8 : e6 15 a5 14 85 37 a5 15  28
9f00 : 85 38 20 44 a6 4c 2d 9e  f1
9f08 : 20 9b b7 c9 2c d0 12 e0  70
9f10 : 02 b0 0e 86 21 20 9b b7  b0
9f18 : a4 21 8a 99 3e 03 4c 37  be
9f20 : 85 4c 48 b2 a2 00 8a 48  19
9f28 : a9 51 a0 83 20 1e ab 68  85
9f30 : aa 48 e0 0a b0 03 20 84  24
9f38 : 94 a9 00 20 cd bd a9 57  c5
9f40 : a0 83 20 1e ab 68 a8 48  9f
9f48 : b9 40 03 aa b9 50 03 20  a2
9f50 : cd bd a9 2d 20 d2 ff 68  75
9f58 : a8 48 b9 60 03 aa b9 70  ec
9f60 : 03 20 cd bd 20 d7 aa 68  db
9f68 : aa e8 e0 10 90 b8 4c 37  2f
9f70 : 85 a9 9f 48 a9 80 48 a9  ce
9f78 : 00 48 48 48 48 4c 31 ea  39
9f80 : a6 c6 f0 1f ca bd 77 02  26
9f88 : c9 85 90 17 c9 8d b0 13  0d
9f90 : 38 e9 85 a8 b9 a6 9f 9d  be
9f98 : 77 02 e8 a9 0d 9d 77 02  1f
9fa0 : e8 86 c6 4c 81 ea 45 58  3c
9fa8 : 4c 49 42 59 4f 40 20 bc  46
9fb0 : f6 20 e1 ff f0 03 4c bc  00
9fb8 : fe 20 15 fd a9 71 a2 9f  bb
9fc0 : 8e 15 03 8d 14 03 ad 00  5b
9fc8 : dd 48 20 a3 fd 68 8d 00  9f
9fd0 : dd ad 18 d0 48 20 a0 e5  78
9fd8 : 68 8d 18 d0 a9 00 8d 91  1b
9fe0 : 02 85 cf a9 48 a2 eb 8e  34
9fe8 : 90 02 8d 8f 02 a9 04 8d  67
9ff0 : 8b 02 a9 0a 8d 89 02 8d  70
9ff8 : 8c 02 20 34 e5 6c 02 a0  1f
```

Listing 1. »Giga Ass« (Schluß)

```
  10;----------------------------------
  11;
  12; ----------------------
  13; hypra-ass quelltext
  14; konvertierungsprogramm
  15; ----------------------
  16;
  17; programmiert von thomas dachsel
  18; am 20. - 21. juni 1987
  19;
  20; dieses programm wandelt einen
  21; hypra-ass quelltext in einen
  22; giga-ass quelltext um, welcher
  23; direkt auf diskette geschrieben
  24; wird.
  25;
  26; starten sie dieses programm mit
  27; <f3>. es werden dann alle not-
  28; wendigen eingaben abgefragt.
  29;
  30;---------------------------------
 100.base $6000
 110.start $6000
 120;
 130; ----------------------------
 140; label- und makro-definitionen
 150; ----------------------------
 160;
 170.equate startadr=$801
 180;
 190.global cnt=count
 200.global ip=inchars
 210.global op=outchars
 220.global inbuf=lib
 230.global outbuf=lob
 240;
 250.macro readchar
 260            jsr get
 270            ldx ip
 280            sta inbuf,x
 290            inc ip
 300.endmacro
 310;
 320.macro key
 330            prn keytx
 340wait        sto 198,0
 350            eqb 198,*
 360            cnq 631,$d,wait
 370            beq leave
 380keytx       .text "←←   <return> druecken"
 390leave       sto 198,0
 400.endmacro
 410;
1000; -------------
1010; hauptprogramm
1020; -------------
1030;
1040; 1. eingabe der file-namen
1050;
1060            jsr clall
1070            prn stx
1080            rfn hqfn
1090            prn ntx
1100            rfn aqfn
1110;
1120; 2. oeffnen der files
1130;
1140            prn o1tx
1150            prn hqfn
1160            prn o2tx
1170            openfile 1,8,0,hqfn
1180            prn o3tx
1190            key
1200            prn o1tx
1210            prn aqfn
1220            prn o4tx
1230            openfile 2,8,1,aqfn
1240            prn o3tx
1250            key
1260            prn mtx
1270            prn aqfn
1280            prn ttx
1290;
1300; 3. initialisiere count
1320;
1330            dst count,startadr
1331            lda #$ff
1332            sta lnr
1333            sta lnr+1
1340;
1350; 4. schreibe neue startadresse
1360;
1370            ldx #2
1380            jsr ckout
1390            out <(startadr)
1400            out >(startadr)
1410;
1420; 5. ueberlies alte startadresse
1430;
1440            ldx #1
1450            jsr chkin
1460            jsr get
1470            jsr get
1480;
1490; 6. erste koppeladresse einlesen
1500;
1501            sto ip,0
1510            readchar
1520            readchar
1530;
1540; 7. zeilennummer (l/h) einlesen
1550;
1560contread    readchar
1570            readchar
1580;
1590; 8. rest der zeile einlesen
1600;
1610loop        readchar
1620            cmp #0
1630            bne loop
1640;
1650; 9. rufe convline auf
1660;
1670            ldx #2
1680            jsr ckout
1690            jsr convline
1700;
1710;10. setze pufferzeiger auf
1720;    pufferanfang zurueck
1730;
1740            sto ip,0
1750;
1760;11. lies koppeladresse der
1770;    naechsten zeile ein
1780;
1790            ldx #1
1800            jsr chkin
1810            readchar
1820            readchar
1830;
1840;12. check ob quelltext-ende
1850;
1860            lda inbuf
1870            bne contread
```

Listing 2. »HYPRA-KONV.SRC« konvertiert Hypra-Ass-Quellcode ins Giga-Ass-Format

```
1880            lda inbuf+1
1890            bne contread
1900;
1910;13. schreibe 00 00 fuer
1920;    quelltext-ende
1930;
1940            ldx #2
1950            jsr ckout
1960            out 0
1970            out 0
1980;
1990;14. schliesse die files, setze
2000;    ein-/ausgabe zurueck
2010;
2020            lda #1
2030            jsr close
2040            lda #2
2050            jsr close
2060            jmp clrch
2070;
2080; ----------------------
2090; ende des hauptprogramms
2100; ----------------------
2110;
3000; ----------------
3010; mnemonics - texte
3020; ----------------
3030;
3040mnemonics .text "cpxcpyldxldycmpadcanddeceorincldaaslbitlsr
3050            .text "orarolrorsbcstastxstyjmpjsrtxataxtyataytsx
3060            .text "txsphpplpphaplabrkrtirtsnopclcseccliseiclv
3070            .text "cldseddeyinydexinxbplbmibvcbvsbccbcsbnebeq
3080;
3090; ---------------------
3100; pseudo-befehls - texte
3110; ---------------------
3120;
3130pseudos   .text "..martgleqbywodstxobbaco
3140            .text "ongoifeleisylienst
3150endpseudos
4000.macro gtc
4010            ldx ip
4020            lda inbuf,x
4030            inc ip
4040.endmacro
4050.macro ptc
4060            ldx op
4070            sta outbuf,x
4080            inc op
4090            icd cnt
4100.endmacro
4110.macro xfer
4120            gtc
4130            ptc
4140.endmacro
4150convline  sto ip,2
4160            sto op,2
4170;
4180; zeilennummer transferieren
4190;
4200            xfer
4210            xfer
4220;
4230; erstes zeichen der zeile:
4240; semikolon, dann kommentarzeile
4250; punkt, dann pseudo
4260; space, dann ueberlesen
4270;        und zum 2. feld gehen
4280; sonst: label ueberlesen
4290;
4300            gtc
4310            cbn 59,nocomm
4320            ptc
4330            jmp rlop
4340nocomm      cbn ".",nopsd
4350            jmp ipsc
4360nopsd       ptc
4370            cbe 32,secfield
4380            cbn 0,*+7      ;end of
4390            jmp endline    ;line ?
4400labelloop xfer
4410            cbn 0,*+7      ;end of
4420            jmp endline    ;line ?
4430            cbn 32,labelloop
4440;
4450; zweites feld:
4460; eventuell label, aber sicher
4470; ein space bereits uebertragen
4480;
4490; jetzt entweder pseudo oder
4500; mnemonic
4510;
4520secfield  gtc
4530            cbn ".",*+7
4540            jmp ipsc
4550            ldx ip         ;x zeigt
4560            ldy #0         ;auf 2.
4570            sty mc         ;zeichen
4580search      lda inbuf-1,x
4590            cmp mnemonics,y
4600            bne nextone
4610            lda inbuf,x
4620            cmp mnemonics+1,y
4630            bne nextone
4640            lda inbuf+1,x
4650            cmp mnemonics+2,y
4660            beq found
4670nextone     iny
4680            iny
4690            iny
4700            .inc mc
4710            cpy #pseudos-mnemonics
4720            bcc search
4730;
4740; mnemonic nicht gefunden:
4750; fehlerhafte zeile, wird ganz
4760; ueberlesen
4770;
4780            jmp rlop
4790found       lda mc         ;mnemonic-
4800            ora #$c0       ;code ab-
4810            ptc            ;legen
4820            gtc            ;3 zeichen
4830            gtc            ;skippen
4840            jmp rlop
5000;
5010; -----------------
5020; insert pseudo-code
5030; -----------------
5040; ip zeigt auf das erste der
5050; beiden pseudo-befehlszeichen
5060;
5070ipsc        ldy #0
5080            ldx ip
5090comploop    lda inbuf,x
5100            cmp pseudos,y
5110            bne nexttext
5120            lda inbuf+1,x
5130            cmp pseudos+1,y
5140            beq foundpsd
5150nexttext    iny
5160            iny
5170            cpy #endpseudos-pseudos
5180            bcc comploop
5190;
5200; pseudo nicht gefunden:
5210; fehlerhafte zeile, wird
5220; ganz ueberlesen
5230;
5240notfound
5250            lda #"."
5260            ptc
5270            jmp rlop
5280foundpsd    gtc
5290            gtc
5300            tya
5310            lsr            ;pseudo-
5320            ora #$a0       ;code in a
5330;
5340; nur speichern, falls kein
5350; macro call -- ausser der makro-
5360; name beginnt mit mnemonic
5370;
5380            cbn $a0,storepsc
5390            ldx ip
5400            ldy #0
5410occurschk   lda inbuf,x
5420            beq storepsc-3
5430            cmp mnemonics,y
5440            bne nextmnemo
5450            lda inbuf+1,x
5460            cmp mnemonics+1,y
5470            bne nextmnemo
5480            lda inbuf+2,x
5490            cmp mnemonics+2,y
5500            bne nextmnemo
5510            lda #$a0
5520            bne storepsc
5530nextmnemo   iny
5540            iny
5550            iny
5560            cpy #pseudos-mnemonics
5570            bcc occurschk
5580            jmp 11
5590storepsc    ptc
5600;
5610; falls .text-pseudo, eventuelles
5620; abschliessendes hochkomma ent-
5630; fernen
5640;
5650            cbn $a8,skip
5660tl1         gtc
5670            cbn 0,*+7      ;end of
5680            jmp put0       ;line ?
5690            ptc
5700            cbn 34,tl1
5710tl2         gtc
5720            cbn 0,*+7      ;end of
5730            jmp put0       ;line ?
5740            cbn 34,*+7
5750            jmp rlop
5760            ptc
5770            jmp tl2
5780;
5790; falls makro-definition oder
5800; -aufruf, klammern wandeln
5810;
5820; es wird die erste klammer auf
5830; in $a0 (shift space) gewandelt
5840; und die letzte klammer zu
5850; geloescht (syntaktisch richtige
5860; klammerung der makro-parameter
5870; vorausgesetzt)
5880;
5890skip        cmp #$a2
5900            bcs rlop
591011          gtc
5920            cbn 0,*+7
5930            jmp put0
5940            cbe "(",12
5950            ptc
5960            jmp 11
```

```
597012          lda #$a0        ;1. "("
5980            ptc             ; --> $a0
5990            sto mc,0
600013          gtc
6010            cbe 0,put0
6020            cbn "(",*+9
6030            inc mc
6040            bpl *+11
6050            cbn ")",*+9
6060            dec mc          ;delete
6070            bmi rlop        ;last ")"
6080            ptc
6090            jmp 13
9000;
9010; transfer rest of line
9020;
9030rlop        xfer
9040            cmp #0
9050            bne rlop
9060;
9070; fuege null-byte an
9080;
9090            lda #0
9100put0        ptc
9110;
9120; speichere koppeladresse
9130;
9140endline     icd count
9150            mvd count,outbuf
9160;
9170; check ob zeile schon einmal
9180; uebertragen
9190;
9200            lda outbuf+2
9210            cmp lnr
9220            bne sendline
9230            lda outbuf+3
9240            cmp lnr+1
9250            bne sendline
9260            rts
9270sendline    mvd outbuf+2,lnr
9280;
9290; schicke zeile zur floppy
9300;
9310            ldx #0
9320sendlop     lda outbuf,x
9330            jsr chrout
9340            inx             ;4 bytes
9350            cpx #5          ;always
9360            bcc sendlop
9370            cmp #0
9380            bne sendlop
9390            rts
10000; -----
10010; texte
10020; -----
10030;
10040stx        .byte $93
10050           .text "←←      konvertierungsprogramm
10060           .text "←←      zum wandeln eines hypra-ass
10070           .text "←←      quelltextes in das giga-ass format
10080           .text "←←←      legen sie die diskette mit dem
10090           .text "←     hypra-ass quelltext in das laufwerk
10100           .text "←     und geben sie den filenamen des
10110           .text "←     hypra-ass quelltextes ein!←←      "
10120ntx        .text "←←      geben sie nun bitte den namen
10130           .text "←     fuer den giga-ass quelltext ein.←←       "
10140o1tx       .byte $93
10150           .text "←←      es wird versucht, das file←←       "
10160o2tx       .text "←←      zum lesen zu oeffnen.←←      "
10170o3tx       .text "←←      das oeffnen war erfolgreich."
10180o4tx       .text "←←      zum schreiben zu oeffnen.←←      "
10190mtx        .byte $93
10200           .text "←←      der angegebene quelltext wird nun
10210           .text "←←      in einen giga-ass quelltext mit←←
                                dem namen "
10220ttx        .text "←←      umgewandelt.←←"
20000;
20010; ------------
20020; datenbereich
20030; ------------
20040;
20041count      .ds 2
20042lnr        .ds 2
20043mc         .ds 1
20050hqfn       .ds 21
20060aqfn       .ds 21
20070inchars    .ds 1
20080lib        .ds 80
20081outchars   .ds 1
20082lob        .ds 80
50000;---------------------------------
50010;
50020; merge file #1: kernel jump table
50030;
50040;---------------------------------
50050.global acptr=$ffa5
50060.global ciout=$ffa8
50070.global untalk=$ffab
50080.global unlisten=$ffae
50090.global listen=$ffb1
50100.global talk=$ffb4
50110.global status=$ffb7
50120.global setlfs=$ffba
50130.global setnam=$ffbd
50140.global open=$ffc0
50150.global close=$ffc3
50160.global chkin=$ffc6
50170.global ckout=$ffc9
50180.global clrch=$ffcc
50190.global basin=$ffcf
50200.global chrout=$ffd2
50210.global load=$ffd5
50220.global save=$ffd8
50230.global settim=$ffdb
50240.global gettim=$ffde
50250.global stopkey=$ffe1
50260.global get=$ffe4
50270.global clall=$ffe7
50280.global udtim=$ffea
50290.global screen=$ffed
50300.global cursor=$fff0
51000;---------------------------------
51010;
51020; merge file #2: low-level macros
51030;
51040;---------------------------------
51050.macro sto a,v
51060           lda #v
51070           sta a
51080.endmacro
51090.macro dst a,dv
51100           lda #<(dv)
51110           sta a
51120           lda #>(dv)
51130           sta a+1
51140.endmacro
51150.macro mov f,t
51160           lda f
51170           sta t
51180.endmacro
51190.macro mdi a,p
51200           lda a
51210           sta &p,y
51220.endmacro
51230.macro mid p,a
51240           lda &p,y
51250           sta a
51260.endmacro
51270.macro mvd f,t
51280           lda f
51290           ldx f+1
51300           sta t
51310           stx t+1
51320.endmacro
51330.macro cbe v,a
51340           cmp #v
51350           beq a
51360.endmacro
51370.macro cbn v,a
51380           cmp #v
51390           bne a
51400.endmacro
51410.macro cnq a,v,b
51420           lda a
51430           cmp #v
51440           bne b
51450.endmacro
51460.macro eqb a,b
51470           lda a
51480           beq b
51490.endmacro
51500.macro nqb a,b
51510           lda a
51520           bne b
51530.endmacro
51540.macro ldw a
51550           lda a
51560           ldy a+1
51570.endmacro
51580.macro stw a
51590           sta a
51600           sty a+1
51610.endmacro
51620.macro icd a
51630           inc a
51640           bne end
51650           inc a+1
51660end
51670.endmacro
51680.macro psh a
51690           lda a
51700           pha
51710.endmacro
51720.macro pll a
51730           pla
51740           sta a
51750.endmacro
51760.macro phx
51770           txa
51780           pha
51790.endmacro
51800.macro plx
51810           pla
51820           tax
51830.endmacro
51840.macro phy
51850           tya
51860           pha
51870.endmacro
51880.macro ply
51890           pla
51900           tay
51910.endmacro
51920.macro prn t
51930           lda #<(t)
51940           ldy #>(t)
51950           jsr $ab1e
```

Listing 2. »HYPRA-KONV.SRC« (Fortsetzung)

```
51960.endmacro
51970.macro out a
51980           lda #a
51990           jsr $ffd2
52000.endmacro
52010.macro mby nr,code
52020           .equate c=n
52030           .on c=0,520
52040           .byte code
52050           .equate c=c-1
52060           .goto 52030
52070.endmacro
55000;---------------------------------
55010;
55020; merge file #3: high-level macros
55030;
55040;---------------------------------
55050;*********************************
55060;
55070; relative load (rld)
55080; -------------
55090; ein file wird relativ in den
55100; speicher ab adresse "adr" ge-
55110; laden. der name des files muss
55120; ab der adresse "filename"
55130; im speicher stehen.
55140; hinter dem letzten zeichen des
55150; namens muss ein nullbyte folgen.
55160;
55170; benoetigt das merge-file #1.
55180;
55190;*********************************
55200.macro rld filename,adr
55210           lda #1        ;lfn
55220           ldx #8        ;dev
55230           ldy #0        ;sa
55240           jsr setlfs
55250;
55260; laenge des filenamens bestimmen
55270;
55280           ldx #0
55290testchar   lda filename,x
55300           beq setlen
55310           inx
55320           bne testchar
55330setlen     txa
55340           ldx #<(filename)
55350           ldy #>(filename)
55360           jsr setnam
55370           lda #0
55380           ldx #<(adr)
55390           ldy #>(adr)
55400           jmp load
55410.endmacro
55420;*********************************
55430;
55440; read filename (rfn)
55450; -------------
55460; ein filename wird von der tasta-
55470; tur eingelesen und ab der adres-
55480; se "adr" in den speicher abge-
55490; legt.
55500; hinter dem letzten zeichen des
55510; namens wird ein nullbyte ange-
55520; fuegt.
55530;
55540; benoetigt merge-files #1 und #2.
55550;
55560;*********************************
55570.macro rfn adr
55580;
55590; tastatur-file oeffnen
55600;
55610           lda #"t"      ;lfn
55620           ldx #0        ;dev
55630           ldy #0        ;sa
55640           jsr setlfs
55650           jsr open
55660           ldx #"t"
55670           jsr chkin
55680;
55690; bildschirm-file oeffnen
55700;
55710           lda #"b"      ;lfn
55720           ldx #3        ;dev
55730           ldy #0        ;sa
55740           jsr setlfs
55750           jsr open
55760           ldx #"b"
55770           jsr ckout
55780;
55790; prompt und input-zeile ausgeben
55800;
55810           prn prompt
55820           jmp getline
55830prompt     .text "filename?
55840           .byte 32
55850           mby 16,$a4
55860           mby 16,$9d
55870revchar    .byte $12,$20,$92,$9d,0
55880delchar    .byte $a4,$9d,$9d,0
55890;
55900; maximal 16 zeichen holen
55910;
55920getline    sto chars,0
55930getchar    jsr get
55940           beq *-3
55950           and #$7f
55960           cbn 20,checkkey
55970           ldx chars
55980           beq getchar
55990           cpx #16
56000           bcc *+5
56010           dec chars
56020           dec chars
56030           prn delchar
56040           prn revchar
56050           jmp getchar
56060checkkey   cmp #32
56070           bcc nonprint
56080           ldx chars
56090           cpx #16
56100           bcc *+3
56110           dex
56120           sta adr,x
56130           jsr chrout
56140           ldx chars
56150           cpx #16
56160           bcs *+5
56170           inc chars
56180           cpx #15
56190           bcs lastchar
56200           prn revchar
56210           jmp getchar
56220lastchar   out $9d
56230           jmp getchar
56240nonprint   cmp #$d
56250           bne getchar
56260           ldx chars
56270           cpx #16
56280           bcs *+7
56290           out $a4
56300           lda #0
56310           sta adr,x
56320;
56330; files schliessen
56340;
56350           lda #"t"
56360           jsr close
56370           lda #"b"
56380           jsr close
56390           jmp *+4
56400chars      .ds 1
56410.endmacro
56420;*********************************
56430;
56440; get disk status (gds)
56450; ---------------
56460;
56470; dieser makro holt den disk-
56480; status in den speicherbereich,
56490; der durch adresse "adr" spezi-
56500; fiziert ist. hinter das letzte
56510; zeichen wird ein nullbyte
56520; abgespeichert.
56530; ausserdem wird noch die nummer
56540; der fehlermeldung in den akku
56550; geholt und das zero-flag ge-
56560; setzt, falls diese 00 war.
56570;
56580; benoetigt das merge-file #1.
56590;
56600;*********************************
56610.macro gds adr
56620           ldx #0
56630           stx chars
56640           sto $ba,8
56650           jsr talk
56660           sto $b9,$6f
56670           jsr $ff96
56680getchars   jsr acptr
56690           ldx chars
56700           sta adr,x
56710           inc chars
56720           cmp #$d
56730           bne getchars
56740           jsr untalk
56750           ldx chars
56760           lda #0
56770           sta adr,x
56780           ldx #0
56790           lda adr
56800           and #%1111
56810           tay
56820           php
56830           lda #0
56840           plp
56850           beq *+8
56860           clc
56870           adc #10
56880           dey
56890           bne *-4
56900           sta chars
56910           lda adr+1
56920           and #%1111
56930           adc chars
56940           cmp #00
56950           jmp *+4
56960chars      .ds 1
56970.endmacro
56980;*********************************
56990;
57000; openfile (opf)
57010; --------
57020;
57030; open <lfn>,<dev>,<sa>,
57040;            "<(fnadr)>,<p/s>,<r/w>"
57050;
57060; in abhaengigkeit von <sa> werden
57070; folgende suffixe an den file-
57080; namen angehaengt:
57090;
```

```
57100; <sa> = 0: ",p,r"
57110; <sa> = 1: ",p,w"
57120; <sa> = 2: ",s,r"
57130; <sa> = 3: ",s,w"
57140;
57150;**********************************
57160.macro openfile lfn,dev,sa,fnadr
57170.if sa=0
57180.equate suffix=prsf
57190.endif
57200.if sa=1
57210.equate suffix=pwsf
57220.endif
57230.if sa=2
57240.equate suffix=srsf
57250.endif
57260.if sa=3
57270.equate suffix=swsf
57280.endif
57290          ldx #0
57300          lda fnadr,x
57310          beq *+5
57320          inx
57330          bne *-6
57340          ldy #0
57350          lda suffix,y
57360          sta fnadr,x
57370          beq *+6
57380          inx
57390          iny
57400          bne *-10
57410          phx              ;length
57420;
57430; setzen der parameter;
57440; aufruf des open-befehls
57450;
57460          lda #lfn
57470          ldx #dev
57480          ldy #sa
57490          jsr setlfs
57500          pla              ;length
57510          ldx #<(fnadr)
57520          ldy #>(fnadr)
57530          jsr setnam
57540          jsr open
57550          bcc openok
57560          prn openerror
57570          rts
57580;
57590; holen des disk-status. falls
57600; ungleich 0, ausstieg!
57610;
57620openok    gds dsbuf
57630          php
57640          prn dsbuf
57650          plp
57660          beq continue
57670          jmp clall
57680prsf      .text ",p,r"
57690pwsf      .text ",p,w"
57700srsf      .text ",s,r"
57710swsf      .text ",s,w"
57720openerror .text "open-befehl meldet fehler!!!←"
57730dsbuf     .ds 40
57740continue
57750.endmacro
```

Listing 2. »HYPRA-KONVERT.SRC« (Schluß)

```
Name : hypra-convert       0801 1417
-------------------------------------
0801 : 0b 08 c1 07 9e 32 30 36   0a
0809 : 31 00 00 00 a9 2c a0 08   c9
0811 : 85 5f 84 60 a9 17 a0 14   71
0819 : 85 5a 84 5b a9 eb a0 6b   ab
0821 : 85 58 84 59 20 bf a3 4c   46
0829 : 00 60 00 20 e7 ff a9 dc   3c
0831 : a0 68 20 1e ab a9 54 a2   70
0839 : 00 a0 00 20 ba ff 20 c0   3b
0841 : ff a2 54 20 c6 ff a9 42   42
0849 : a2 03 a0 00 20 ba ff 20   ad
0851 : c0 ff a2 42 20 c9 ff a9   a5
0859 : 36 a0 60 20 1e ab 4c 69   3f
0861 : 60 46 49 4c 45 4e 41 4d   27
0869 : 45 3f 20 a4 a4 a4 a4 a4   36
0871 : a4 a4 a4 a4 a4 a4 a4 a4   70
0879 : a4 a4 a4 9d 9d 9d 9d 9d   c5
0881 : 9d 9d 9d 9d 9d 9d 9d 9d   80
0889 : 9d 9d 9d 12 20 92 9d 00   ac
0891 : a4 9d 9d 00 a9 00 8d ed   18
0899 : 60 20 e4 ff f0 fb 29 7f   d5
08a1 : c9 14 d0 20 ae ed 60 f0   6a
08a9 : f0 e0 10 90 03 ce ed 60   3f
08b1 : ce ed 60 a9 65 a0 60 20   e0
08b9 : 1e ab a9 60 a0 60 20 1e   ed
08c1 : ab 4c 6e 60 c9 20 90 2e   76
08c9 : ae ed 60 e0 10 90 01 ca   c1
08d1 : 9d 1d 6b 20 d2 ff ae ed   9f
08d9 : 60 e0 10 b0 03 ee ed 60   e4
08e1 : e0 0f b0 0a a9 60 a0 60   97
08e9 : 20 1e ab 4c 6e 60 a9 9d   58
08f1 : 20 d2 ff 4c 6e 60 c9 0d   2f
08f9 : d0 9f ae ed 60 e0 10 b0   b1
0901 : 05 a9 a4 20 d2 ff a9 00   dc
0909 : 9d 1d 6b a9 54 20 c3 ff   9a
0911 : a9 42 20 c3 ff 4c ee 60   3b
0919 : 00 a9 d1 a0 69 20 1e ab   de
0921 : a9 54 a2 00 a0 00 20 ba   9d
0929 : ff 20 c0 ff a2 54 20 c6   43
0931 : ff a9 42 a2 03 a0 00 20   5f
0939 : ba ff 20 c0 ff a2 42 20   71
0941 : c9 ff a9 21 a0 61 20 1e   6a
0949 : ab 4c 54 61 46 49 4c 45   c6
0951 : 4e 41 4d 45 3f 20 a4 a4   0d
0959 : a4 a4 a4 a4 a4 a4 a4 a4   58
0961 : a4 a4 a4 a4 a4 a4 9d 9d   36
0969 : 9d 9d 9d 9d 9d 9d 9d 9d   68
0971 : 9d 9d 9d 9d 9d 9d 12 20   47
0979 : 92 9d 00 a4 9d 9d 00 a9   89
0981 : 00 8d d8 61 20 e4 ff f0   b5
0989 : fb 29 7f c9 14 d0 20 ae   d7
0991 : d8 61 f0 f0 e0 10 90 03   4b
0999 : ce d8 61 ce d8 61 a9 50   e5
09a1 : a0 61 20 1e ab a9 4b a0   34
09a9 : 61 20 1e ab 4c 59 61 c9   c0
09b1 : 20 90 2e ae d8 61 e0 10   b7
09b9 : 90 01 ca 9d 32 6b 20 d2   d5
09c1 : ff ae d8 61 e0 10 b0 03   d1
09c9 : ee d8 61 e0 0f b0 0a a9   8a
09d1 : 4b a0 61 20 1e ab 4c 59   ec
09d9 : 61 a9 9d 20 d2 ff 4c 59   8b
09e1 : 61 c9 0d d0 9f ae d8 61   1a
09e9 : e0 10 b0 05 a9 a4 20 d2   84
09f1 : ff a9 00 9d 32 6b [illegible] 54   46
09f9 : 20 c3 ff a9 42 20 c3 ff   64
0a01 : 4c d9 61 00 a9 1d a0 6a   6d
0a09 : 20 1e ab a9 1d a0 6b 20   1d
0a11 : 1e ab a9 43 a0 6a 20 1e   f2
0a19 : ab a2 00 bd 1d 6b f0 03   c4
0a21 : e8 d0 f8 a0 00 b9 87 62   74
0a29 : 9d 1d 6b f0 04 e8 c8 d0   9a
0a31 : f4 8a 48 a9 01 a2 08 a0   38
0a39 : 00 20 ba ff 68 a2 1d a0   49
0a41 : 6b 20 bd ff 20 c0 ff 90   55
0a49 : 08 a9 9b a0 62 20 1e ab   18
0a51 : 60 a2 00 8e 78 62 a9 08   26
0a59 : 85 ba 20 b4 ff a9 6f 85   f0
0a61 : b9 20 96 ff 20 a5 ff ae   5c
0a69 : 78 62 9d b9 62 ee 78 62   f5
0a71 : c9 0d d0 f0 20 ab ff ae   d0
0a79 : 78 62 a9 00 9d b9 62 a2   03
0a81 : 00 ad b9 62 29 0f a8 08   d0
0a89 : a9 00 28 f0 06 18 69 0a   35
0a91 : 88 d0 fa 8d 78 62 ad ba   b8
0a99 : 62 29 0f 6d 78 62 c9 00   c3
0aa1 : 4c 79 62 00 08 a9 b9 a0   38
0aa9 : 62 20 1e ab 28 f0 5d 4c   30
0ab1 : e7 ff 2c 50 2c 52 00 2c   5b
0ab9 : 50 2c 57 00 2c 53 2c 52   a8
0ac1 : 00 2c 53 2c 57 00 4f 50   85
0ac9 : 45 4e 2d 42 45 46 45 48   f5
0ad1 : 4c 20 4d 45 4c 44 45 54   ce
0ad9 : 20 46 45 48 4c 45 52 21   f1
0ae1 : 21 21 0d 00 00 00 00 00   d6
0ae9 : 00 00 00 00 00 00 00 00   ea
0af1 : 00 00 00 00 00 00 00 00   f2
0af9 : 00 00 00 00 00 00 00 00   fa
0b01 : 00 00 00 00 00 00 00 00   02
0b09 : 00 00 00 00 a9 63 a0 6a   16
0b11 : 20 1e ab a9 00 a0 63 20   33
0b19 : 1e ab a9 00 85 c6 a5 c6   2a
0b21 : f0 fc ad 77 02 c9 0d d0   2e
0b29 : f1 f0 17 0d 0d 20 20 20   8c
0b31 : 3c 52 45 54 55 52 4e 3e   10
0b39 : 20 44 52 55 45 43 4b 45   e1
0b41 : 4e 00 a9 00 85 c6 a9 1d   69
0b49 : a0 6a 20 1e ab a9 32 a0   fc
0b51 : 6b 20 1e ab a9 85 a0 6a   e7
0b59 : 20 1e ab a2 00 bd 32 6b   55
0b61 : f0 03 e8 d0 f8 a0 00 b9   2f
0b69 : ce 63 9d 32 6b f0 04 e8   b7
0b71 : c8 d0 f4 8a 48 a9 02 a2   4f
0b79 : 08 a0 01 20 ba ff 68 a2   a8
0b81 : 32 a0 6b 20 bd ff 20 c0   c0
0b89 : ff 90 08 a9 dd a0 63 20   b8
0b91 : 1e ab 60 a2 00 8e ba 63   17
0b99 : a9 08 85 ba 20 b4 ff a9   fa
0ba1 : 6f 85 b9 20 96 ff 20 a5   7a
0ba9 : ff ae ba 63 9d fb 63 ee   3f
0bb1 : ba 63 c9 0d d0 f0 20 ab   9d
0bb9 : ff ae ba 63 a9 00 9d fb   33
0bc1 : 63 a2 00 ad fb 63 29 0f   c9
0bc9 : a8 08 a9 00 28 f0 06 18   32
0bd1 : 69 0a 88 d0 fa 8d ba 63   49
0bd9 : ad fc 63 29 0f 6d ba 63   10
0be1 : c9 00 4c bb 63 00 08 a9   df
0be9 : fb a0 63 20 1e ab 28 f0   d3
0bf1 : 5d 4c e7 ff 2c 50 2c 52   09
0bf9 : 00 2c 50 2c 57 00 2c 53   76
0c01 : 2c 52 00 2c 53 2c 57 00   d0
0c09 : 4f 50 45 4e 2d 42 45 46   22
0c11 : 45 48 4c 20 4d 45 4c 44   4a
0c19 : 45 54 20 46 45 48 4c 45   ac
0c21 : 52 21 21 21 0d 00 00 00   41
0c29 : 00 00 00 00 00 00 00 00   2a
0c31 : 00 00 00 00 00 00 00 00   32
0c39 : 00 00 00 00 00 00 00 00   3a
0c41 : 00 00 00 00 00 00 00 00   42
0c49 : 00 00 00 00 00 00 a9 63   b7
0c51 : a0 6a 20 1e ab a9 42 a0   45
0c59 : 64 20 1e ab a9 00 85 c6   09
0c61 : a5 c6 f0 fc ad 77 02 c9   77
0c69 : 0d d0 f1 f0 17 0d 0d 20   c7
0c71 : 20 20 3c 52 45 54 55 52   ec
0c79 : 4e 3e 20 44 52 55 45 43   e2
0c81 : 4b 45 4e 00 a9 00 85 c6   41
0c89 : a9 a9 a0 6a 20 1e ab a9   71
0c91 : 32 a0 6b 20 1e ab a9 04   e0
0c99 : a0 6b 20 1e ab a9 01 8d   e2
0ca1 : 18 6b a9 08 8d 19 6b a9   7d
0ca9 : ff 8d 1a 6b 8d 1b 6b a2   07
0cb1 : 02 20 c9 ff a9 01 20 d2   fe
0cb9 : ff a9 08 20 d2 ff a2 01   4d
0cc1 : 20 c6 ff 20 e4 ff 20 e4   e1
0cc9 : ff a9 00 8d 47 6b 20 e4   69
0cd1 : ff ae 47 6b 9d 48 6b ee   0e
0cd9 : 47 6b 20 e4 ff ae 47 6b   e4
0ce1 : 9d 48 6b ee 47 6b 20 e4   75
0ce9 : ff ae 47 6b 9d 48 6b ee   26
0cf1 : 47 6b 20 e4 ff ae 47 6b   fc
0cf9 : 9d 48 6b ee 47 6b 20 e4   8d
0d01 : ff ae 47 6b 9d 48 6b ee   3e
0d09 : 47 6b c9 00 d0 f0 a2 02   9b
0d11 : 20 c9 ff 20 05 66 a9 00   44
```

Listing 3. »Hypra-Konvert« bitte mit dem MSE (siehe Seite 159) eingeben und mit RUN starten

```
0d19 : 8d 47 6b a2 01 20 c6 ff  a5
0d21 : 20 e4 ff ae 47 6b 9d 48  60
0d29 : 6b ee 47 6b 20 e4 ff ae  d1
0d31 : 47 6b 9d 48 6b ee 47 6b  c0
0d39 : ad 48 6b d0 a9 ad 49 6b  03
0d41 : d0 a4 a2 02 20 c9 ff a9  f0
0d49 : 00 20 d2 ff a9 00 20 d2  cf
0d51 : ff a9 01 20 c3 ff a9 02  50
0d59 : 20 c3 ff 4c cc ff 43 50  5f
0d61 : 58 43 50 59 4c 44 58 4c  7b
0d69 : 44 59 43 4d 50 41 44 43  7b
0d71 : 41 4e 44 44 45 43 45 4f  95
0d79 : 52 49 4e 43 4c 44 41 41  da
0d81 : 53 4c 42 49 54 4c 53 52  4e
0d89 : 4f 52 41 52 4f 4c 52 4f  db
0d91 : 52 53 42 43 53 54 41 53  09
0d99 : 54 58 53 54 59 4a 4d 50  36
0da1 : 4a 53 52 54 58 41 54 41  17
0da9 : 58 54 59 41 54 41 59 54  07
0db1 : 53 58 54 58 53 50 48 50  ca
0db9 : 50 4c 50 50 48 41 50 4c  b6
0dc1 : 41 42 52 4b 52 54 49 52  b3
0dc9 : 54 53 4e 4f 50 43 4c 43  1b
0dd1 : 53 45 43 43 4c 49 53 45  e7
0dd9 : 49 43 4c 56 43 4c 44 53  f0
0de1 : 45 44 44 45 59 49 4e 59  ce
0de9 : 44 45 58 49 4e 58 42 50  60
0df1 : 4c 42 4d 49 42 56 43 42  43
0df9 : 56 53 42 43 43 42 43 53  ec
0e01 : 42 4e 45 42 45 51 2e 2e  f8
0e09 : 4d 41 52 54 47 4c 45 51  a5
0e11 : 42 59 57 4f 44 53 54 58  a0
0e19 : 4f 42 42 41 43 4f 4f 4e  cb
0e21 : 47 4f 49 46 45 4c 45 49  89
0e29 : 53 59 4c 49 45 4e 53 54  22
0e31 : a9 02 8d 47 6b a9 02 8d  4f
0e39 : 98 6b ae 47 6b bd 48 6b  b8
0e41 : ee 47 6b ae 98 6b 9d 99  12
0e49 : 6b ee 98 6b ee 18 6b d0  be
0e51 : 03 ee 19 6b ae 47 6b bd  cd
0e59 : 48 6b ee 47 6b ae 98 6b  61
0e61 : 9d 99 6b ee 98 6b ee 18  54
0e69 : 6b d0 03 ee 19 6b ae 47  11
0e71 : 6b bd 48 6b ee 47 6b c9  a5
0e79 : 3b d0 14 ae 98 6b 9d 99  86
0e81 : 6b ee 98 6b ee 18 6b d0  f6
0e89 : 03 ee 19 6b 4c 68 68 c9  f4
0e91 : 2e d0 03 4c 14 67 ae 98  da
0e99 : 6b 9d 99 6b ee 98 6b ee  e6
0ea1 : 18 6b d0 03 ee 19 6b c9  fc
0ea9 : 20 f0 2c c9 00 d0 03 4c  b1
0eb1 : 99 68 ae 47 6b bd 48 6b  af
0eb9 : ee 47 6b ae 98 6b 9d 99  8a
0ec1 : 6b ee 98 6b ee 18 6b d0  36
0ec9 : 03 ee 19 6b c9 00 d0 03  dd
0ed1 : 4c 99 68 c9 20 d0 db ae  92
0ed9 : 47 6b bd 48 6b ee 47 6b  70
0ee1 : c9 2e d0 03 4c 14 67 ae  b6
0ee9 : 47 6b a0 00 8c 1c 6b bd  e1
0ef1 : 47 6b d9 33 65 d0 10 bd  63
0ef9 : 48 6b d9 34 65 d0 08 bd  6c
0f01 : 49 6b d9 35 65 f0 0d c8  c0
0f09 : c8 c8 ee 1c 6b c0 a8 90  f5
0f11 : de 4c 68 68 ad 1c 6b 09  b8
0f19 : c0 ae 98 6b 9d 99 6b ee  f6
0f21 : 98 6b ee 18 6b d0 03 ee  55
0f29 : 19 6b ae 47 6b bd 48 6b  29
0f31 : ee 47 6b ae 47 6b bd 48  cb
0f39 : 6b ee 47 6b 4c 68 68 a0  46
0f41 : 00 ae 47 6b bd 48 6b d9  57
0f49 : db 65 d0 08 bd 49 6b d9  93
0f51 : dc 65 f0 1c c8 c8 c0 2a  ca
0f59 : 90 ea a9 2e ae 98 6b 9d  27
0f61 : 99 6b ee 98 6b ee 18 6b  e4
0f69 : d0 03 ee 19 6b 4c 68 68  25
0f71 : ae 47 6b bd 48 6b ee 47  7f
0f79 : 6b ae 47 6b bd 48 6b ee  24
0f81 : 47 6b 98 4a 09 a0 c9 a0  eb
0f89 : d0 2d ae 47 6b a0 00 bd  bc
0f91 : 48 6b f0 20 d9 33 65 d0  3d
0f99 : 14 bd 49 6b d9 34 65 d0  c2
0fa1 : 0c bd 4a 6b d9 35 65 d0  0a
0fa9 : 04 a9 a0 d0 0a c8 c8 c8  60
0fb1 : c0 a8 90 db 4c f5 67 ae  d4
0fb9 : 98 6b 9d 99 6b ee 98 6b  09
0fc1 : ee 18 6b d0 03 ee 19 6b  93
0fc9 : c9 a8 d0 50 ae 47 6b bd  73
0fd1 : 48 6b ee 47 6b c9 00 d0  1a
0fd9 : 03 4c 88 68 ae 98 6b 9d  ca
0fe1 : 99 6b ee 98 6b ee 18 6b  64
0fe9 : d0 03 ee 19 6b c9 22 d0  49
0ff1 : db ae 47 6b bd 48 6b ee  0c
0ff9 : 47 6b c9 00 d0 03 4c 88  d0
1001 : 68 c9 22 d0 03 4c 68 68  f6
1009 : ae 98 6b 9d 99 6b ee 98  74
1011 : 6b ee 18 6b d0 03 ee 19  7a
1019 : 6b 4c c6 67 c9 a2 b0 73  a4
1021 : ae 47 6b bd 48 6b ee 47  2f
1029 : 6b c9 00 d0 03 4c 88 68  19
1031 : c9 28 f0 14 ae 98 6b 9d  66
1039 : 99 6b ee 98 6b ee 18 6b  bc
1041 : d0 03 ee 19 6b 4c f5 67  31
1049 : a9 a0 ae 98 6b 9d 99 6b  e2
1051 : ee 98 6b ee 18 6b d0 03  6a
1059 : ee 19 6b a9 00 8d 1c 6b  98
1061 : ae 47 6b bd 48 6b ee 47  6f
1069 : 6b c9 00 f0 46 c9 28 d0  cc
1071 : 05 ee 1c 6b 10 09 c9 29  25
1079 : d0 05 ce 1c 6b 30 14 ae  e9
1081 : 98 6b 9d 99 6b ee 98 6b  d1
1089 : ee 18 6b d0 03 ee 19 6b  5b
1091 : 4c 35 68 ae 47 6b bd 48  bf
1099 : 6b ee 47 6b ae 98 6b 9d  53
10a1 : 99 6b ee 98 6b ee 18 6b  24
10a9 : d0 03 ee 19 6b c9 00 d0  80
10b1 : e2 a9 00 ae 98 6b 9d 99  cc
10b9 : 6b ee 98 6b ee 18 6b d0  2e
10c1 : 03 ee 19 6b ee 18 6b d0  ee
10c9 : 03 ee 19 6b ad 18 6b ae  9e
10d1 : 19 6b 8d 99 6b 8e 9a 6b  a3
10d9 : ad 9b 6b cd 1a 6b d0 09  3b
10e1 : ad 9c 6b cd 1b 6b d0 01  c3
10e9 : 60 ad 9b 6b ae 9c 6b 8d  0d
10f1 : 1a 6b 8e 1b 6b a2 00 bd  0f
10f9 : 99 6b 20 d2 ff e8 e0 05  7f
1101 : 90 f5 c9 00 d0 f1 60 93  43
1109 : 0d 0d 20 20 20 4b 4f 4e  df
1111 : 56 45 52 54 49 45 52 55  dc
1119 : 4e 47 53 50 52 4f 47 52  4b
1121 : 41 4d 4d 0d 0d 20 20 20  90
1129 : 5a 55 4d 20 57 41 4e 44  c6
1131 : 45 4c 4e 20 45 49 4e 45  96
1139 : 53 20 48 59 50 52 41 2d  d1
1141 : 41 53 53 0d 0d 20 20 20  35
1149 : 51 55 45 4c 4c 54 45 58  4d
1151 : 54 45 53 20 49 4e 20 44  31
1159 : 41 53 20 47 49 47 41 2d  63
1161 : 41 53 53 20 46 4f 52 4d  e7
1169 : 41 54 0d 0d 0d 20 20 20  4c
1171 : 4c 45 47 45 4e 20 53 49  a0
1179 : 45 20 44 49 45 20 44 49  02
1181 : 53 4b 45 54 54 45 20 4d  e0
1189 : 49 54 20 44 45 4d 0d 20  c0
1191 : 20 20 48 59 50 52 41 2d  f6
1199 : 41 53 53 20 51 55 45 4c  ca
11a1 : 4c 54 45 58 54 20 49 4e  7c
11a9 : 20 44 41 53 20 4c 41 55  ba
11b1 : 46 57 45 52 4b 0d 20 20  1c
11b9 : 20 55 4e 44 20 47 45 42  76
11c1 : 45 4e 20 53 49 45 20 44  68
11c9 : 45 4e 20 46 49 4c 45 4e  af
11d1 : 41 4d 45 4e 20 44 45 53  b4
11d9 : 0d 20 20 20 48 59 50 52  38
11e1 : 41 2d 41 53 53 20 51 55  9a
11e9 : 45 4c 4c 54 45 58 54 45  e5
11f1 : 53 20 45 49 4e 21 0d 0d  0b
11f9 : 20 20 20 00 0d 0d 20 20  2b
1201 : 20 47 45 42 45 4e 20 53  4c
1209 : 49 45 20 4e 55 4e 20 42  94
1211 : 49 54 54 45 20 44 45 4e  18
1219 : 20 4e 41 4d 45 4e 0d 20  96
1221 : 20 20 46 55 45 52 20 44  7e
1229 : 45 4e 20 47 49 47 41 2d  b5
1231 : 41 53 53 20 51 55 45 4c  62
1239 : 4c 54 45 58 54 20 45 49  fa
1241 : 4e 2e 0d 0d 20 20 20 00  0f
1249 : 93 0d 0d 20 20 20 45 53  69
1251 : 20 57 49 52 44 20 56 45  e3
1259 : 52 53 55 43 48 54 2c 20  2b
1261 : 44 41 53 20 46 49 4c 45  89
1269 : 0d 0d 20 20 20 00 0d 0d  59
1271 : 20 20 20 5a 55 4d 20 4c  ce
1279 : 45 53 45 4e 20 5a 55 20  ed
1281 : 4f 45 46 46 4e 45 4e 2e  72
1289 : 0d 0d 20 20 20 00 0d 0d  79
1291 : 20 20 20 44 41 53 20 4f  20
1299 : 45 46 46 4e 45 4e 20 57  53
12a1 : 41 52 20 45 52 46 4f 4c  e9
12a9 : 47 52 45 49 43 48 2e 00  c3
12b1 : 0d 0d 20 20 20 5a 55 4d  16
12b9 : 20 53 43 48 52 45 49 42  56
12c1 : 45 4e 20 5a 55 20 4f 45  9f
12c9 : 46 46 4e 45 4e 2e 0d 0d  13
12d1 : 20 20 20 00 93 0d 0d 20  20
12d9 : 20 20 44 45 52 20 41 4e  8b
12e1 : 47 45 47 45 42 45 4e 45  57
12e9 : 20 51 55 45 4c 4c 54 45  b3
12f1 : 58 54 20 57 49 52 44 20  df
12f9 : 4e 55 4e 0d 0d 20 20 20  ba
1301 : 49 4e 20 45 49 4e 45 4e  db
1309 : 20 47 49 47 41 2d 41 53  31
1311 : 53 20 51 55 45 4c 4c 54  04
1319 : 45 58 54 20 4d 49 54 0d  2e
1321 : 0d 20 20 20 44 45 4d 20  2e
1329 : 4e 41 4d 45 4e 20 00 0d  14
1331 : 0d 20 20 20 55 4d 47 45  c2
1339 : 57 41 4e 44 45 4c 54 2e  b1
1341 : 0d 0d 00 00 00 00 00 00  d5
1349 : 00 00 00 00 00 00 00 00  4a
1351 : 00 00 00 00 00 00 00 00  52
1359 : 00 00 00 00 00 00 00 00  5a
1361 : 00 00 00 00 00 00 00 00  62
1369 : 00 00 00 00 00 00 00 00  6a
1371 : 00 00 00 00 00 00 00 00  72
1379 : 00 00 00 00 00 00 00 00  7a
1381 : 00 00 00 00 00 00 00 00  82
1389 : 00 00 00 00 00 00 00 00  8a
1391 : 00 00 00 00 00 00 00 00  92
1399 : 00 00 00 00 00 00 00 00  9a
13a1 : 00 00 00 00 00 00 00 00  a2
13a9 : 00 00 00 00 00 00 00 00  aa
13b1 : 00 00 00 00 00 00 00 00  b2
13b9 : 00 00 00 00 00 00 00 00  ba
13c1 : 00 00 00 00 00 00 00 00  c2
13c9 : 00 00 00 00 00 00 00 00  ca
13d1 : 00 00 00 00 00 00 00 00  d2
13d9 : 00 00 00 00 00 00 00 00  da
13e1 : 00 00 00 00 00 00 00 00  e2
13e9 : 00 00 00 00 00 00 00 00  ea
13f1 : 00 00 00 00 00 00 00 00  f2
13f9 : 00 00 00 00 00 00 00 00  fa
1401 : 00 00 00 00 00 00 00 00  02
1409 : 00 00 00 00 00 00 00 00  0a
1411 : 00 00 00 00 08 00 00 ff  92
```

Listing 3. (Schluß)

```
 10;********************************
 11;
 12; bildschirmspeicher verschieben
 13; ------------------------------
 14;
 15; giga-ass hilfs-routine
 16;
 17; der bildschirmspeicher wird
 18; von $0400-$07ff nach $cc00-
 19; $cfff verschoben.
 20;
 21; danach ist der quelltextspeicher
 22; in $0400-$8000 (31 k).
 23;
 24;********************************
100.object "init 31 k,p,w"
110.base $2c3
120.global p=$fc
130;
140; 1. rom-zeichensatz in ram
150;    umkopieren ($d000-$dfff)
160;
170start      ldy #0
180           sty p
190           lda #$d0
200           sta p+1
210           sei
220loop       lda #$33
230           sta 1
240           lda &p,y
250           sty 1
260           sta &p,y
270           inc p
280           bne loop
290           inc p+1
300           ldx p+1
310           cpx #$e0
320           bcc loop
330           lda #$37
340           sta 1
350;
360; 2. alten bildschirm-ram freigeben
370;
380           lda #4
390           sta $282
400;
410; 3. bildschirmspeicher auf $cc00
420;
430           lda $dd00
440           and #%11111100
450           sta $dd00
460           lda #$35
470           sta $d018
480           lda #$cc
490           sta $288
500;
510; 4. giga-ass re-initialisieren
520;
530           jmp ($8009)
540;
550; 5. auto-start adresse setzen
560;
570           .word 0,start
```

Listing 4. »SCOURCE-INIT« verschiebt den Bildschirm nach $CC00 und schafft so ein weiteres KByte Speicherplatz für Quellcode

Paradoxon-Basic: neue Befehle und mehr Speicherplatz

Durch die Verwendung von Basic-Erweiterungen wird der Arbeitsspeicher meist eingeschränkt. Mit Paradoxon-Basic ist das anders. Neben 14 neuen Befehlen stehen nun 12 KByte RAM mehr für die Basic-Programmierung zur Verfügung.

Darauf haben schon viele gewartet: Eine Programmierhilfe, die vor allem das Schreiben umfangreicher Programme erleichtert und bereits existierende Befehle verbessert, um Ihnen umständliche Programmiertricks zu ersparen.

Der Programmname »Paradaxon-Basic« (Listing 1) kommt daher, daß es paradox erscheint, wenn sich Basic plus Erweiterung im RAM unter dem Kernel befinden, das heißt, gleicher Adreßraum, aber verschiedene Ebenen. Normalerweise ist im C64 der Bereich von $0800 bis $9FFF als Basic-Speicher vorgesehen. Im Bereich von $A000 bis $FFFF bleiben so jedoch 24 KByte für Basic ungenutzt, da an diesen Stellen der Basic-Interpreter, der Input-Output-Bereich und das Kernel eingeblendet werden.

Der erweiterte Basic-Speicherplatz wird nun dadurch gewonnen, daß der Basic-Interpreter unter den Input-Output-Bereich und das Kernel verschoben wird. Dabei bleiben noch rund 3 KByte frei, die für die Befehlserweiterung und die neuen Umschaltroutinen genutzt werden.

12 KByte mehr

Folgende Speicherbereiche müssen immer gleichzeitig zur Verfügung stehen:
1. Basic-RAM + Basic-Interpreter, oder
2. Basic-RAM + I/O-Bereich + Kernel.

Da der I/O-Bereich nicht verschoben werden kann, diente er als Ansatzpunkt für die Speicheraufteilung. Der Basic-Interpreter mußte daher unter den I/O-Bereich und das Kernel:
1. Basic-RAM plus Basic-Interpreter $0800 bis $CFFF $D000 bis $F4C0
2. Basic-RAM plus I/O-Bereich plus Kernel $0800 bis $CFFF, $D000 bis $DFFF und $E000 bis $FFFF. In Bild 1 haben wir dies noch einmal grafisch veranschaulicht.

Da die Kernel-Routinen für die Arbeit des Basic-Interpreters unverzichtbar sind, waren Umschaltroutinen

```
Name : paradoxon-basic   0801 158b
-----------------------------------
0801 : 22 08 c3 07 9e 32 34 35  b0
0809 : 34 3a 12 20 50 41 52 41  be
0811 : 44 4f 58 4f 4e 20 42 41  6e
0819 : 53 49 43 20 35 30 4b 20  28
0821 : 00 00 00 00 00 00 00 6d  fc
0829 : 2d 0e 2c cd ec cc ce 4d  2d
0831 : ee 4c 20 ad ae ac 4e 0d  a7
0839 : 2e 6e ed 8d 8e 8c 7d 3d  89
0841 : 1e dd de 5d fe bd bc 5e  3f
0849 : 1d 3e 7e fd 9d 79 39 d9  23
0851 : 59 b9 be 19 f9 99 6c 69  4a
0859 : 29 c9 e0 c0 49 a9 a2 a0  65
0861 : 09 e9 65 25 06 24 c5 e4  bf
0869 : c4 c6 45 e6 a5 a6 a4 46  6d
0871 : 05 26 66 e5 85 86 84 75  69
0879 : 35 16 d5 d6 55 f6 b5 b4  57
0881 : 56 15 36 76 f5 95 94 b6  8a
0889 : 96 90 b0 f0 30 d0 10 50  1c
0891 : 70 71 31 d1 51 b1 11 f1  0b
0899 : 91 61 21 c1 41 a1 01 e1  44
08a1 : 81 00 93 0d 20 20 20 20  6d
08a9 : 20 20 20 2a 2a 2a 2a 20  04
08b1 : 50 41 52 41 44 4f 58 4f  1d
08b9 : 4e 20 42 41 53 49 43 20  9d
08c1 : 2a 2a 2a 2a 0d 0d 20 35  f4
08c9 : 30 4b 20 52 41 4d 20 53  97
08d1 : 59 53 54 45 4d 20 20 00  e8
08d9 : 52 52 42 59 1e fe 47 fe  05
08e1 : 04 fe 00 85 62 84 64 a0  32
08e9 : 00 84 61 84 63 b1 61 91  81
08f1 : 63 c8 d0 f9 e6 62 e6 64  11
08f9 : e4 62 d0 f1 60 85 62 8e  59
0901 : 65 00 b1 61 c9 e0 b0 06  71
0909 : c9 a0 90 06 69 20 69 0f  63
0911 : 91 61 c8 c8 ea c4 65 90  29
0919 : e9 60 85 62 86 64 b1 61  f5
0921 : c9 a9 d0 15 20 5c 09 20  df
0929 : 5c 09 b1 61 c9 a0 d0 09  99
0931 : 20 5c 09 20 6e 09 20 5c  2e
0939 : 09 a2 7a b1 61 dd 27 08  1a
0941 : f0 09 ca d0 f8 20 5c 09  97
0949 : 4c 1f 09 20 5c 09 e0 31  5f
0951 : b0 f3 20 5c 09 20 6e 09  ec
0959 : 4c 46 09 c8 d0 02 e6 62  a1
0961 : a5 64 c5 62 d0 06 c4 63  0d
0969 : d0 02 68 68 60 b1 61 29  cd
0971 : f0 c9 e0 f0 0d c9 90 90  1e
0979 : 0f c9 c0 b0 0b b1 61 69  4a
0981 : 21 2c b1 61 69 0f 91 61  69
0989 : 60 a2 27 99 00 ff c8 c8  ec
0991 : c8 ca d0 f7 60 78 a9 37  d0
0999 : 85 01 a9 a0 a8 a2 c0 20  00
09a1 : e4 08 a9 e0 a0 f0 a2 e5  f8
09a9 : 20 e4 08 a9 34 85 01 a9  39
09b1 : a0 a0 d0 a2 c0 20 e4 08  da
09b9 : a9 d0 a0 01 a2 80 20 fe  bf
09c1 : 08 a9 d3 a0 29 a2 64 20  20
09c9 : fe 08 a9 f4 a0 48 a2 53  52
09d1 : 20 fe 08 a9 c8 8d 15 09  07
09d9 : a9 d0 a0 82 a2 9e 20 fe  00
09e1 : 08 a9 3d 85 63 a9 d3 a0  d2
09e9 : 8a a2 f3 20 1b 09 a9 5e  23
09f1 : 85 63 a9 f3 a0 7b a2 f4  6b
09f9 : 20 1b 09 a2 35 bd a3 08  1d
0a01 : 9d 73 f4 ca 10 f7 a9 0a  6a
0a09 : a2 c8 86 61 85 62 a9 f4  d9
0a11 : a2 c0 86 63 85 64 a0 00  1f
0a19 : b1 61 91 63 c8 d0 f9 e6  14
0a21 : 62 e6 64 a5 64 c9 ff d0  fa
0a29 : ef a9 20 a0 81 20 8a 09  5e
0a31 : a9 eb a0 82 20 8a 09 a9  16
0a39 : fd a0 83 20 8a 09 a2 09  f9
0a41 : bd d9 08 9d f6 ff ca d0  dd
0a49 : f7 a2 00 a0 00 bd 67 0a  45
0a51 : f0 11 85 62 e8 bd 67 0a  a6
0a59 : 85 61 e8 bd 67 0a 91 61  50
0a61 : e8 d0 ea 6c fc ff e8 16  99
0a69 : 20 e8 17 d7 e8 18 fd e8  d7
0a71 : 19 ea e8 38 20 e8 3f d7  36
0a79 : e8 40 fd e8 41 ea e8 47  bb
0a81 : f6 e8 2a 4c e8 2b e2 e8  45
0a89 : 2c fd f1 31 f1 df 1c d0  86
0a91 : df 24 f3 d9 dd ef d8 a1  be
0a99 : c0 d8 a2 f4 d0 24 c5 d0  f3
0aa1 : 25 f4 da b6 e1 da b7 f4  8b
0aa9 : e3 b4 20 e3 b5 f5 e2 4b  98
0ab1 : 77 e2 4c f5 f4 4f c0 f4  22
0ab9 : 50 f5 d4 97 81 d4 98 fb  45
0ac1 : f1 da 08 f1 dc 01 00 20  76
0ac9 : 8a dd 4c f7 e7 d0 03 4c  fd
0ad1 : 1d d8 20 8a dd 20 f7 e7  42
0ad9 : 20 13 d6 a4 60 a6 5f d0  27
0ae1 : 01 88 ca 86 41 84 42 60  ac
0ae9 : c9 28 f0 03 4c 9e dd 20  d4
0af1 : 9b e7 e0 19 b0 29 86 d6  f7
0af9 : 20 79 00 c9 2c d0 23 20  25
0b01 : 9b e7 e0 28 b0 19 86 d3  63
0b09 : 20 79 00 c9 29 d0 13 18  b5
0b11 : a6 d6 a4 d3 20 f0 ff 68  20
0b19 : 68 20 73 00 4c a2 da a2  f9
0b21 : 0e 2c a2 0b 6c 00 03 a9  75
0b29 : 46 a0 00 d1 7a f0 03 4c  cd
0b31 : e1 e3 20 73 00 b0 eb e9  83
0b39 : 2f f0 e4 c9 09 b0 e0 4a  81
0b41 : b0 02 69 03 0a 0a 0a 0a  da
0b49 : 0a 85 a7 a9 fc 85 a8 20  14
0b51 : 73 00 a9 b2 20 ff de 20  43
0b59 : 9e dd 24 0d 10 c4 20 a6  86
0b61 : e6 aa a0 00 b1 22 91 a7  86
0b69 : f0 0e c8 98 38 65 a7 f0  d5
0b71 : f5 a9 00 ca f0 f0 d0 ec  48
0b79 : 68 68 60 ea ea 00 e0 13  e3
0b81 : f0 03 4c 37 d4 a0 02 b1  ab
0b89 : 5f 18 65 5f 85 61 c8 b1  23
0b91 : 5f 65 60 85 62 88 a5 31  cf
0b99 : 38 f1 5f 85 31 c8 a5 32  a7
0ba1 : f1 5f 85 32 a0 00 b1 61  7d
0ba9 : 91 5f c8 d0 f9 e6 60 e6  5c
0bb1 : 62 a5 32 c5 60 b0 ef a5  c2
0bb9 : 31 85 5f a5 32 85 60 4c  a3
0bc1 : 61 e2 a2 09 6c 00 03 20  70
0bc9 : 73 00 c9 40 f0 09 20 79  81
0bd1 : 00 20 ed d7 4c ae d7 a2  36
0bd9 : 08 20 73 00 b0 03 20 9e  af
0be1 : e7 e0 08 90 dd e0 0c b0  c3
0be9 : d9 a9 0f a0 6f 20 ba ff  52
0bf1 : 20 79 00 f0 2a c9 24 f0  4f
0bf9 : 6a a5 7a a4 7b 20 87 e4  0a
0c01 : 20 e2 e7 20 a3 e6 a6 22  e0
0c09 : a4 23 20 bd ff 20 c0 ff  03
0c11 : b0 44 20 cc ff a5 b8 20  d5
0c19 : c3 ff b0 44 4c ae d7 a9  7d
0c21 : 00 20 bd ff 20 c0 ff b0  0a
0c29 : 2d a9 00 85 90 a5 ba 20  3d
0c31 : b1 ff a5 b9 09 f0 20 93  42
0c39 : ff a5 90 30 17 a6 b8 20  ff
0c41 : c6 ff b0 12 20 cf ff b0  57
0c49 : 0d 20 d2 ff b0 08 c9 0d  a7
0c51 : d0 f2 f0 be a9 05 48 20  d2
0c59 : cc ff a5 b8 20 c3 ff 68  96
0c61 : 4c f9 f0 20 73 00 f0 05  ef
```

Listing 1. »Paradoxon-Basic« bitte mit dem MSE (Seite 159) eingeben

```
0c69 : a2 0b 6c 00 03 a9 24 85  c5
0c71 : 61 a9 01 a2 60 86 b9 e8  2e
0c79 : a0 00 20 bd ff 20 c0 ff  dd
0c81 : b0 d4 a6 b8 20 c6 ff b0  f6
0c89 : cd 20 c1 f6 20 c1 f6 a9  f5
0c91 : 0d 20 d2 ff 20 e1 ff f0  56
0c99 : b9 a5 c6 f0 08 c6 c6 a5  12
0ca1 : c6 f0 fc c6 c6 20 c1 f6  5a
0ca9 : 20 c1 f6 20 c1 f6 aa 20  2a
0cb1 : c1 f6 20 cd ed a9 20 20  9c
0cb9 : d2 ff b0 9a 20 c1 f6 f0  d8
0cc1 : ce 20 d2 ff 90 f6 b0 8e  f4
0cc9 : 20 cf ff b0 89 24 90 70  c4
0cd1 : 02 a8 60 68 68 4c 0b f6  4f
0cd9 : ea 20 d4 f1 38 a5 2d e9  80
0ce1 : 02 aa a5 2e e9 00 a8 a9  fc
0ce9 : 00 85 b9 20 d5 ff b0 09  50
0cf1 : a5 90 29 bf d0 06 4c a7  de
0cf9 : f1 4c f9 f0 a2 1d 2c a2  b6
0d01 : 09 2c a2 0b 6c 00 03 f0  df
0d09 : f9 90 05 c9 ab d0 f3 38  46
0d11 : 20 6b d9 20 13 d6 a5 5f  9e
0d19 : 85 61 a5 60 85 62 20 79  a3
0d21 : 00 f0 df c9 ab d0 db 20  bb
0d29 : 73 00 90 0c f0 02 b0 d2  c9
0d31 : a9 ff 85 14 85 15 d0 09  14
0d39 : 20 6b d9 e6 14 d0 02 e6  00
0d41 : 15 20 13 d6 38 a5 5f e5  00
0d49 : 61 a5 60 e5 62 90 b3 60  8c
0d51 : 20 00 f7 a0 00 b1 5f 91  b1
0d59 : 61 c8 d0 f9 e6 60 e6 62  63
0d61 : a5 2e c5 60 b0 ef 90 05  72
0d69 : a9 01 a8 91 2b 20 33 d5  1b
0d71 : 18 a5 22 69 02 85 2d a5  5e
0d79 : 23 69 00 85 2e 20 59 d6  f9
0d81 : 4c 74 d4 20 79 00 f0 2d  f6
0d89 : b0 75 20 6b d9 a5 14 85  8f
0d91 : a9 85 ab a5 15 85 aa 85  d0
0d99 : ac 20 79 00 c9 2c d0 5f  b4
0da1 : 20 73 00 b0 5a 20 6b d9  99
0da9 : a5 14 85 ad a5 15 85 ae  e6
0db1 : 05 14 f0 48 60 a9 00 85  64
0db9 : aa 85 ac 85 ae a9 64 85  d6
0dc1 : a9 85 ab a9 0a 85 ad 60  91
0dc9 : a9 2b 85 a7 a9 00 85 a8  60
0dd1 : 60 a0 00 b1 a7 aa c8 b1  0e
0dd9 : a7 86 a7 85 a8 b1 a7 60  d5
0de1 : a5 a9 85 ab a5 aa 85 ac  51
0de9 : 60 18 a5 ab 65 ad 85 ab  65
0df1 : a5 ac 65 ae 85 ac b0 04  a4
0df9 : c9 fa 90 ec a2 0e 2c a2  92
0e01 : 0b 6c 00 03 20 7c f7 20  a9
0e09 : c1 f7 20 ca f7 f0 26 20  07
0e11 : e2 f7 90 f6 20 d9 f7 20  e3
0e19 : c1 f7 20 ca f7 f0 10 a0  c0
0e21 : 03 a5 ac 91 a7 88 a5 ab  01
0e29 : 91 a7 20 e2 f7 90 eb 20  e6
0e31 : 5e d6 4c 74 d4 20 c1 f7  e1
0e39 : 20 ca f7 f0 d7 a0 03 c8  fa
0e41 : b1 a7 f0 f4 30 0f c9 22  87
0e49 : d0 f5 c8 b1 a7 f0 e9 c9  b9
0e51 : 22 d0 f7 f0 ea c9 8f f0  14
0e59 : df c9 89 f0 0c c9 8d f0  c4
0e61 : 08 c9 a7 f0 04 c9 8c d0  b8
0e69 : d6 18 98 65 a7 85 7a 85  ba
0e71 : af a5 a8 69 00 85 7b 85  6f
0e79 : b0 20 73 00 90 12 aa f0  3c
0e81 : b7 c9 2c d0 ba a5 7a 85  10
0e89 : af a5 7b 85 b0 4c 72 f8  c3
0e91 : 20 6b d9 20 d9 f7 a5 2b  2b
0e99 : 85 64 a5 2c 85 65 a0 03  4b
0ea1 : b1 64 88 c5 15 d0 06 b1  b2
0ea9 : 64 c5 14 f0 19 20 e2 f7  21
0eb1 : 88 b1 64 aa 88 b1 64 85  33
0eb9 : 64 86 65 c8 b1 64 d0 de  12
0ec1 : a9 f9 a2 ff d0 04 a5 ac  2c
0ec9 : a6 ab 85 62 86 63 a2 90  22
0ed1 : 38 20 49 ec 20 df ed a5  0d
0ed9 : af 85 7a a5 b0 85 7b a2  08
0ee1 : 00 a0 00 20 73 00 90 08  bf
0ee9 : b9 00 01 f0 15 20 07 f9  63
0ef1 : b9 00 01 f0 05 81 7a c8  e1
0ef9 : d0 e9 20 39 f9 20 79 00  74
0f01 : 90 f8 38 a5 7a e5 a7 a8  97
0f09 : 20 79 00 4c 75 f8 98 48  82
0f11 : 38 a5 2e 85 62 85 64 e5  08
0f19 : 7b aa e8 a4 7a 84 61 c8  9b
0f21 : 84 63 d0 02 e6 64 a0 ff  df
0f29 : b1 61 91 63 88 d0 f9 b1  b6
0f31 : 61 91 63 c6 62 c6 64 ca  90
0f39 : d0 ec a9 30 81 7a d0 2a  73
0f41 : 98 48 a5 7b 85 62 85 64  20
0f49 : a4 7a d0 02 c6 64 84 61  03
0f51 : 88 84 63 38 a5 2e e5 7b  56
0f59 : aa e8 a0 01 b1 61 91 63  f3
0f61 : c8 d0 f9 e6 62 e6 64 ca  71
0f69 : d0 f2 20 33 d5 18 a5 22  1a
0f71 : 69 02 85 2d a5 23 69 00  fb
0f79 : 85 2e a2 00 68 a8 60 20  4c
0f81 : 7c f7 a9 88 8d 9d d4 a9  e1
0f89 : f9 8d 9e d4 4c a3 f9 20  95
0f91 : 6b d9 d0 0b a9 6b 8d 9d  e6
0f99 : d4 a9 d9 8d 9e d4 60 a5  c7
0fa1 : 14 85 ab a5 15 85 ac 20  88
0fa9 : e2 f7 a5 ac 85 62 a5 ab  df
0fb1 : 85 63 a2 90 38 20 49 ec  26
0fb9 : 20 df ed a2 00 e8 bd ff  d7
0fc1 : 00 9d 76 02 d0 f7 a9 20  21
0fc9 : 9d 76 02 86 c6 60 a0 02  e9
0fd1 : 84 7b a0 00 84 7a a2 20  22
0fd9 : b1 a9 10 01 e8 29 7f 91  7c
0fe1 : 7a c8 b1 a9 10 04 e8 e8  f7
0fe9 : e8 e8 29 7f d0 02 a9 20  84
0ff1 : 91 7a c8 8a 91 7a c8 c9  e6
0ff9 : 21 60 a9 00 91 7a a8 b1  a8
1001 : 7a f0 06 20 0c f1 c8 d0  8e
1009 : f6 a9 3d 20 0c f1 20 b5  63
1011 : da 20 2c d8 20 e4 ff f0  2c
1019 : 05 20 e4 ff f0 fb 60 a2  1d
1021 : 0b 2c a2 0e 6c 00 03 d0  21
1029 : f6 a9 0d 20 0c f1 a6 2e  82
1031 : a5 2d 86 aa 85 a9 e4 30  fd
1039 : d0 04 c5 2f f0 14 20 c7  22
1041 : f9 f0 03 20 f3 f9 a6 aa  76
1049 : a5 a9 18 69 07 90 e3 e8  4c
1051 : d0 e0 60 d0 ca a9 0d 20  32
1059 : 0c f1 a6 30 a5 2f 86 aa  51
1061 : 85 a9 e4 32 d0 04 c5 31  e1
1069 : f0 e8 20 c7 f9 a9 28 91  7f
1071 : 7a c8 b1 a9 aa 86 ab e0  40
1079 : 08 b0 a7 c8 b1 a9 9d 30  1b
1081 : fd c8 b1 a9 9d 38 fd de  d5
1089 : 38 fd c8 ca d0 ee a4 ab  ba
1091 : a9 00 99 40 fd 99 48 fd  72
1099 : 88 10 f7 a0 02 a2 00 84  79
10a1 : 7b 86 7a c8 e8 8c a7 00  a8
10a9 : 8e a8 00 bd 48 fd 85 63  94
10b1 : bd 40 fd 85 62 38 a2 90  52
10b9 : 20 49 ec 20 dd ed ac a7  0c
10c1 : 00 a2 ff c8 e8 bd 01 01  ae
10c9 : 91 7a d0 f7 ae a8 00 e4  c4
10d1 : ab f0 06 a9 2c 91 7a d0  86
10d9 : cb a9 29 91 7a c8 20 f3  4c
10e1 : f9 a6 ab bd 40 fd dd 30  9c
10e9 : fd d0 15 bd 48 fd dd 38  a8
10f1 : fd d0 0d a9 00 9d 40 fd  b9
10f9 : 9d 48 fd ca d0 e5 f0 0a  a7
1101 : fe 48 fd d0 96 fe 40 fd  1b
1109 : d0 91 a0 03 b1 a9 aa 88  4e
1111 : b1 a9 18 65 a9 a8 8a 65  1e
1119 : aa aa 98 4c 57 fa c9 22  81
1121 : d0 03 20 73 00 20 c1 f7  e1
1129 : 20 ca f7 d0 01 60 a0 03  62
1131 : a6 7a c8 b1 a7 f0 f1 dd  02
1139 : 00 02 d0 f6 84 a9 c8 e8  d8
1141 : b1 a7 f0 05 dd 00 02 f0  6a
1149 : f5 a4 a9 bd 00 02 d0 e0  c7
1151 : a5 a7 85 5f a5 a8 85 60  8e
1159 : a0 02 b1 5f 85 14 c8 b1  d2
1161 : 5f 85 15 a9 cd 8d d1 d6  3c
1169 : a9 60 8d 12 d7 8d 14 d7  d2
1171 : 20 c9 d6 a9 4c 8d 14 d7  92
1179 : a9 d0 8d 12 d7 a9 20 8d  96
1181 : d1 d6 20 0a fa 4c 21 fb  95
1189 : a2 00 86 61 a0 ff e8 c8  38
1191 : b1 7a f0 19 30 17 5d c9  a3
1199 : fb f0 f3 c9 80 f0 11 e8  e8
11a1 : bd c9 fb 10 fa e6 61 a5  fb
11a9 : 61 c9 08 d0 df 4c 79 d5  fd
11b1 : 68 68 20 b1 fb 4c 74 d4  29
11b9 : 18 c8 98 65 7a 85 7a a5  11
11c1 : 61 0a aa bd f1 fb 48 bd  25
11c9 : f0 fb 48 20 79 d5 4c 73  2b
11d1 : 00 4d 45 52 47 c5 44 45  52
11d9 : 4c 45 54 c5 4f 4c c4 52  a5
11e1 : 45 4e 55 4d 42 45 d2 41  68
11e9 : 55 54 cf 56 41 d2 41 52  7b
11f1 : 52 41 d9 46 49 4e c4 d1  e1
11f9 : f6 48 f7 60 f7 fc f7 77  53
1201 : f9 1f fa 4b fa 16 fb 40  83
1209 : 24 0d 00 00 00 00 00 00  b4
1211 : 00 00 00 00 00 00 00 00  12
1219 : 00 00 00 00 00 00 00 00  1a
1221 : 00 00 00 00 00 00 00 4c  ba
1229 : 49 53 54 0d 00 00 00 00  d3
1231 : 00 00 00 00 00 00 00 00  32
1239 : 00 00 00 00 00 00 00 00  3a
1241 : 00 00 00 00 00 00 00 52  e6
1249 : 55 4e 0d 00 00 00 00 00  09
1251 : 00 00 00 00 00 00 00 00  52
1259 : 00 00 00 00 00 00 00 00  5a
1261 : 00 00 00 00 00 00 00 4c  fa
1269 : 4f 41 44 20 1d 1d 1d 1d  d7
1271 : 1d 1d 1d 1d 1d 1d 1d 1d  71
1279 : 1d 1d 1d 1d 1d 1d 3a 0d  ce
1281 : 00 00 00 00 00 00 00 41  04
1289 : 4e 4c 45 49 54 55 4e 47  30
1291 : 20 49 4e 20 44 45 52 20  e5
1299 : 36 34 27 45 52 8d 00 00  ed
12a1 : 00 00 00 00 00 00 00 00  a2
12a9 : 00 00 00 00 00 00 00 00  aa
12b1 : 00 00 00 00 00 00 00 00  b2
12b9 : 00 00 00 00 00 00 00 00  ba
12c1 : 00 00 00 00 00 00 00 00  c2
12c9 : 00 00 00 00 00 00 00 00  ca
12d1 : 00 00 00 00 00 00 00 00  d2
12d9 : 00 00 00 00 00 00 00 00  da
12e1 : 00 00 00 00 00 00 00 00  e2
12e9 : 00 00 00 00 00 00 00 00  ea
12f1 : 00 00 00 00 00 00 00 00  f2
12f9 : 00 00 00 00 00 00 00 00  fa
1301 : 00 00 00 00 00 00 00 a5  4d
1309 : fe f0 27 10 0a a9 05 c5  d9
1311 : cd b0 1f 85 cd 90 1b a6  ca
1319 : c6 ec 89 02 b0 14 ac 2e  b3
1321 : fd b9 00 fc f0 0a 9d 77  5f
1329 : 02 ee 2e fd e6 c6 d0 e7  a5
1331 : 85 fe 4c ee 02 c0 00 2a  a1
1339 : 2a 2a 2a 2a 2a 2a 2a 2a  39
1341 : 2a 2a 2a 2a 2a 2a 2a 2a  41
1349 : 2a 2a 2a 2a 2a 2a 2a 2a  49
1351 : 2a 2a 2a 2a 2a 2a 2a a5  48
1359 : fe f0 2a 10 7e e4 c6 d0  28
1361 : 7a a6 c6 f0 76 bd 76 02  31
1369 : c9 85 90 08 c9 8d b0 04  ee
1371 : a9 00 85 fe ac 2e fd 99  c3
1379 : 00 fc ee 2e fd c8 cc 2f  30
1381 : fd f0 ed d0 56 e0 ff f0  da
1389 : 27 a6 c6 f0 4e bd 76 02  84
1391 : c9 85 90 47 c9 8d b0 43  7c
1399 : e9 83 85 fe c6 c6 38 e9  dc
13a1 : 01 0a 0a 0a 0a 0a 8d 2e  ef
13a9 : fd 69 1f 8d 2f fd d0 2b  51
13b1 : ad 8d 02 29 04 f0 24 a5  6e
13b9 : cb c9 07 b0 1e c9 03 90  9e
13c1 : 1a d0 02 a9 07 e9 03 a8  16
13c9 : ad 8d 02 29 01 f0 04 c8  1c
13d1 : c8 c8 c8 98 09 80 85 fe  eb
13d9 : 98 d0 c3 4c fb 02 78 a0  47
13e1 : 00 e6 01 b1 14 c6 01 58  f7
13e9 : 60 78 e6 01 91 14 c6 01  36
13f1 : 58 60 08 85 fb 68 85 fc  3f
13f9 : 68 38 e9 02 8d ae 02 68  5f
1401 : 8d af 02 a5 fc 48 a5 fb  3c
1409 : 4c a7 02 8d fe 02 8e ff  95
1411 : 02 a9 02 48 a9 bf 48 ba  a1
1419 : bd 03 01 48 ae ff 02 ad  ef
1421 : fe 02 4c b8 02 78 85 fd  40
1429 : a9 02 48 a9 bf 48 08 a5  c4
1431 : fd 4c c5 02 20 ea ff a5  ab
1439 : 91 c9 db d0 03 6c fc ff  47
1441 : 20 e1 ff f0 03 4c e2 02  92
1449 : 20 b5 fe 6c 02 d0 78 a9  6d
1451 : 47 8d fa ff a2 ff 9a d8  64
1459 : a9 35 85 01 a9 05 8d 16  43
1461 : d0 c6 01 a9 00 a8 99 02  ba
1469 : 00 99 00 02 99 00 03 c8  ad
1471 : d0 f4 a9 01 8d a6 02 a2  a1
1479 : 3c a9 03 86 b2 85 b3 a9  95
1481 : 08 8d 82 02 a9 04 8d 88  33
1489 : 02 85 62 a2 d0 b1 61 49  ed
1491 : ff 91 61 d1 61 d0 0d 49  4f
1499 : ff 91 61 c8 d0 ef e6 62  bf
14a1 : e4 62 d0 e9 a6 62 8c 83  de
14a9 : 02 8e 84 02 20 b5 fe a9  53
14b1 : 08 85 fe a9 e0 8d 2e fd  a0
14b9 : 58 6c 00 d0 20 e1 fe a9  c2
14c1 : 40 8d cc 02 a9 1e 8d fa  f3
14c9 : ff 20 84 ff c6 01 20 81  f1
14d1 : ff e6 01 a9 00 8d 20 d0  47
14d9 : a9 00 8d 21 d0 a9 03 8d  8b
14e1 : 86 02 c6 01 20 e1 fe 60  08
14e9 : a2 56 bd 02 ff 9d a7 02  f5
14f1 : ca 10 f7 a2 26 bd 59 ff  cb
14f9 : 9d 14 03 ca 10 f7 a9 f3  0a
1501 : 8d 8f 02 a9 02 8d 90 02  de
1509 : 60 78 e6 01 e6 01 28 20  d6
1511 : 00 00 08 78 c6 01 c6 01  b4
1519 : 28 60 e6 01 e6 01 6c fe  71
1521 : ff 78 c6 01 c6 01 40 e6  71
1529 : 01 e6 01 6c fa ff 48 8a  51
1531 : 48 98 48 a9 7f 8d 0d dd  61
1539 : ac 0d dd 30 0b c6 01 c6  62
1541 : 01 4c 2d fe e6 01 e6 01  a8
1549 : 4c 72 fe c6 01 4c 00 fd  d5
1551 : e6 01 4c 31 ea 20 48 eb  9a
1559 : c6 01 4c 50 fd e6 01 60  99
1561 : e9 02 34 03 cc 02 4a f3  a7
1569 : 91 f2 0e f2 50 f2 33 f3  a7
1571 : 57 f1 ca f1 ed f6 3e f1  25
1579 : 2f f3 34 03 a5 f4 ed f5  b5
1581 : a9 34 85 01 4c 41 fe 00  91
1589 : 00 00 ff 00 ff 00 ff 00  89
```

Listing 1. (Schluß)

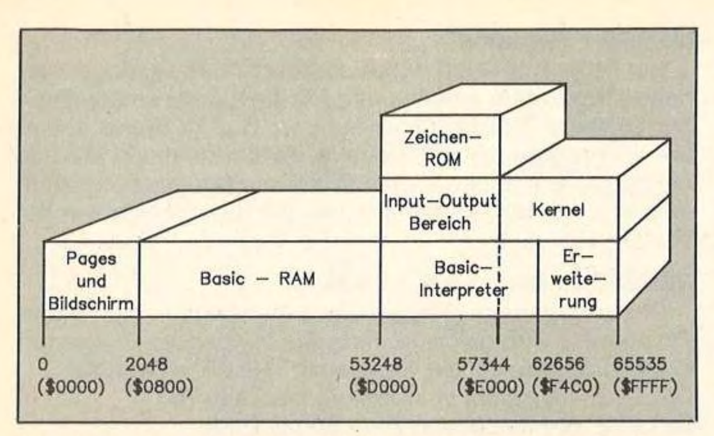


Bild 1. Die neue Speicheraufteilung von Paradoxon Basic: Der Basic-Interpreter wurde verschoben.

notwendig, um zwischen den beiden Speicherkonfigurationen hin- und herzuschalten. Insgesamt werden vier Umschaltroutinen benötigt:
- die Umschaltung zur Nutzung von Kernel-Routinen,
- die Interrupt-Behandlung,
- die NMI-Behandlung,
- die Reset-Behandlung.

Generell gilt: Die Umschaltung zwischen den beiden Speicheraufteilungen darf nur von einem Speicherbereich aus erfolgen, der selbst außerhalb des betroffenen Bereiches liegt, also beispielsweise aus ungenutzten Speicherstellen der ersten Pages. Die Speicherstellen $02A7 bis $02FF sind ungenutzt und können dafür verwendet werden. Alle vier Umschaltroutinen arbeiten nach dem gleichen Prinzip: Sie legen eine Rücksprungadresse auf den Stack, schalten auf das Kernel um und starten die benötigte Routine. Der Rücksprung erfolgt nun auf die Umschaltroutine, die wieder ins RAM verzweigt. Der NMI wird in das RAM umgeleitet, wo auf <RUN/STOP> bzw. <CTRL CBM RESTORE> getestet wird, um gegebenenfalls einen Warmstart oder einen Reset auszuführen.

Die NMI- oder Reset-Routinen im Kernel können wegen der Basic-Verschiebung nicht mehr verwendet werden, denn sie würden abstürzen. Da hinter den Umschaltroutinen und dem Basic-Interpreter noch etwa 2 KByte Speicherplatz überbleiben, wurde hier die neue Befehlserweiterung eingebaut. Sie verbessert auch vorhandene Befehle, enthält Diskettenkommandos, Programmierhilfen und eine Funktionstastenbelegung. Zusammen füllen diese Befehle den gesamten verbliebenen Speicher bis auf ein paar Byte aus.

So wird der gesamte Speicher des C64 optimal genutzt und die Einschaltmeldung verkündet:
51199 BASIC BYTES FREE

Die neue Speicheraufteilung finden Sie in Tabelle 1 noch einmal sehr detailliert. Wen die Funktionsweise des Programms noch genauer interessiert, der findet auf der Programmservice-Diskette zu dieser Ausgabe ein ausführlich dokumentiertes Sourcecode-Listing (4 Vizawrite-Files), das für Einsteiger ebenso interessant sein dürfte wie für Profis.

Viele neue Befehle

Paradoxon-Basic ist mit einem Basic-Lader ausgestattet und wird nach LOAD "PARADAXON-BASIC",8 und RUN installiert. Nach zirka sieben Sekunden ist das Programm generiert und gibt seine Einschaltmeldung aus.

Neben dem erweiterten Basic-Speicherplatz von 50 KByte hat man eine Befehlserweiterung zur Verfügung, die sich in mehrere Teile gliedert:

Die LOAD- und SAVE-Befehle verwenden automatisch die Sekundäradresse 1. Das heißt, es wird – sofern nicht anders angegeben – absolut geladen (LOAD "NAME",8,1).

Durch Drücken von <CTRL CBM RESTORE> wird ein Reset ausgeführt, durch den Paradoxon-Basic neu gestartet wird.

Die Funktionstasten <F1> bis <F8> können jeweils mit bis zu 31 Zeichen Text und Steuerzeichen belegt werden. Um sie zu programmieren, drückt man <CTRL> zusammen mit der entsprechenden Funktionstaste. Daraufhin verdoppelt sich die Blinkgeschwindigkeit des Cursors. Alle Tastatureingaben werden jetzt in den Speicher übernommen. Die Eingabe wird durch Drücken einer beliebigen Funktionstaste oder bei Überschreitung der maximalen Länge beendet.

Programmierbarer Reset

Nach jedem Reset wird die Funktion der Taste <F8> ausgeführt. Es ist also beispielsweise möglich, automatisch ein weiteres Programm in den Computer zu laden und zu starten, indem die entsprechenden Befehle für »F8« eingegeben werden. Sie können auf diese Weise Paradoxon-Basic auch direkt in eigene Programme einbauen.

Bereich			Inhalt
0000	-	07FF	erste Pages und Bildschirm
02A7	-	02FF	Umschaltroutinen
0800	-	CFFF	BASIC-Speicherbereich
D000	-	FFFF	I/O-Bereich und Kernal
D000	-	F4BF	BASIC-Interpreter
F4C0	-	F5BA	eingebundene Befehle
F5BB	-	F6D0	Diskettenbefehle
F6D2	-	FBFF	zusätzliche Befehle
FC00	-	FCFF	Funktionstasten-Belegung
FD00	-	FDD6	Funktionstasten-Routinen
FDD7	-	FFFF	Umschaltroutinen

Tabelle 1. Die Speicheraufteilung von Paradoxon-Basic unterscheidet sich deutlich von der Normalkonfiguration

Einige bereits vorhandene Basic-Befehle sind verbessert worden:

GOTO X

So ist es möglich, nach den Befehlen GOTO und GOSUB einen beliebigen numerischen Ausdruck (Variable oder Formel) anzugeben. Zum Beispiel:
GOTO (A*2)/B.

An der daraus berechneten Zeilennummer wird das Programm dann fortgesetzt.

Um bei einem <RENUMBER> Fehler zu vermeiden, sollte der zu berechnende Ausdruck jedoch nicht mit einer Zahl beginnen. Falsch wäre beispielsweise: GOTO 2*A. Richtig muß es heißen: GOTO A*2.

RESTORE X

Der Zeiger auf das nächste DATA-Element läßt sich auf jede Zeilennummer (X) einstellen, die über einen numerischen Ausdruck vorgegeben wird. Existiert die Zeilennummer nicht, werden die DATAs aus der nächsten Programmzeile gelesen.

PRINT(X,Y)

Nach dem PRINT-Befehl kann man in Klammern die Bildschirmposition angeben, an der der Ausdruck stattfinden soll. Die linke obere Ecke des Bildschirms ist die Position (0,0). Die Maximalwerte für die Cursorposition sind also 39 und 24. Achtung: Ein numerischer Ausdruck nach PRINT darf nicht mit »(« beginnen !

DIM X(N)
Man kann ein Array beliebig oft neu dimensionieren. Das alte Array wird dabei gelöscht und das neue generiert. Wird ein Array nicht mehr gebraucht, wird es mit (0) dimensioniert; es belegt dann fast keinen Speicherplatz mehr.

DEFF1= "..." bis DEFF8= "..."
Die Funktionstasten lassen sich auch vom Programm aus definieren, wobei ein beliebiger String-Ausdruck Verwendung findet. Es lassen sich sogar mehr als 31 Zeichen pro Taste speichern, wenn man dafür eine andere Taste nicht belegt.

Allgemeine Änderungen
1. Paradoxon-Basic laden (nicht starten !) 2. POKE 5333, Rahmenfarbe POKE 5338, Hintergrundfarbe POKE 5343, Zeichenfarbe POKE 2755, Geräteadresse für LOAD ... POKE 2758, Sekundäradresse für LOAD ... POKE 3033, Geräteadresse für Diskettenbefehle POKE 5297, Nummer der Funktionstaste, die nach einem RESET ausgeführt wird (1 bis 8) POKE 5301, Wert dieser Funktionstaste für: F1=0, F3=32, F5=64, F7=96, F2=128, F4=160, F6=192, F8=224 3. Paradoxon-Basic speichern (unter anderem Namen!)
Funktionstastenbelegung
1. Paradoxon-Basic laden und starten 2. Funktionstasten nach Belieben programmieren 3. Paradoxon-Basic erneut laden 4. Gegebenenfalls obige allgemeine Änderungen durchführen 5. FOR I = 0 TO 255 POKE 4616 + I, PEEK (64512 + I) NEXT I in einer Direktmodus-Zeile ausführen. 6. Paradoxon-Basic speichern (unter anderem Namen!)
Arbeiten mit dem 64'er DOS
1. Original-Kernel einsetzen 2. Paradoxon-Basic laden 3. FOR I = 0 TO 1280:POKE 5632 + I, PEEK (57344 + I):NEXT I in einer Direktmodus-Zeile ausführen und POKE 2468, 22 POKE 2472, 27 POKE 45, 192 POKE 46, 26 CLR eingeben. 4. Paradoxon-Basic speichern (anderer Name!) 5. 64'er DOS wieder einbauen

Tabelle 2. Tips für die Anpassung an eigene Bedürfnisse

Diskettenbefehle können sowohl im Direkt-Modus als auch im Programm-Modus verwendet werden. Sie werden alle mit dem Symbol »@« und der Gerätenummer eingeleitet. Wird keine Nummer angegeben, so bezieht sich der Befehl auf die Geräteadresse 8. Es ist allerdings darauf zu achten, daß kein File mit der logischen Filenummer 15 geöffnet ist.

@: Der Fehlerkanal der Floppy wird ausgelesen und auf dem Bildschirm angezeigt.

@$: Das Inhaltsverzeichnis einer Diskette wird auf dem Bildschirm ausgegeben. Dabei kann die Ausgabe durch <RUN/STOP> unterbrochen oder durch Drücken einer beliebigen Taste angehalten werden.

Außerdem kann ein Diskettenbefehl zur Floppy gesendet werden, indem man ihn hinter »@« anhängt.

Die folgenden Befehle können Sie nur im Direktmodus verwenden:

MERGE "Filename"
Ein Programm wird hinter das bereits im Speicher stehende Programm geladen und mit diesem zu einem einzigen größeren Programm verbunden. Das im Speicher stehende Programm muß kleinere Zeilennummern als das nachgeladene haben, da sonst kein lauffähiges Programm entsteht. Für die Gerätenummer gilt das gleiche wie bei LOAD.

DELETE Anfangszeile - Endzeile
Der angegebene Bereich eines im Speicher stehenden Programms wird gelöscht. Falls Sie ein ganzes Programm löschen wollen, sollten Sie jedoch »NEW« verwenden.
DELETE -70 löscht vom Anfang bis Zeile 70
DELETE 80- löscht von Zeile 80 bis Ende

OLD
Ein durch NEW oder Reset gelöschtes Programm wird wieder hergestellt.

Ein mit DELETE gelöschter Bereich beziehungsweise gelöschtes Programm kann leider nicht zurückgeholt werden, da DELETE Speicherbereiche umkopiert und somit endgültig löscht.

RENUMBER Startzeile, Abstand
Die Zeilennummern eines Programms werden entsprechend den Angaben neu numeriert. Dabei werden auch alle Adressen hinter GOTO, GOSUB, THEN, RESTORE, ON GOTO und ON GOSUB aktualisiert. Werden keine Angaben zur Einteilung gemacht, so wird RENUMBER 100,10 angenommen.

AUTO Startzeile, Abstand
Automatische Zeilennumerierung bei der Programmeingabe. Werden keine weiteren Werte angegeben, wird mit Zeile 100 begonnen, und in Zehnerschritten weitergezählt. Man kann während der Eingabe eine bereits eingegebene Zeile korrigieren, ohne die AUTO-Funktion abzubrechen, da die nächste Zeilennummer stets aus der zuletzt ausgegebenen berechnet wird. Unterbrochen wird die AUTO-Funktion, wenn nach einer Zeilennummer <RETURN> eingegeben wird.

VAR
Dieser Befehl gibt alle bisherigen Variablen in der Reihenfolge ihrer Definition auf dem Bildschirm aus. Die Ausgabe kann mit <RUN/STOP> unterbrochen beziehungsweise durch Drücken einer beliebigen Taste angehalten werden. Mit OPEN 1,4:CMD 1:VAR wird die Ausgabe auf den Drucker geleitet.

ARRAY
listet sämtliche Arrays mit ihren Inhalten auf. Arrays mit mehr als sieben Dimensionen werden nicht ausgegeben. Falls man solche Arrays benötigt, sollten diese ganz am Ende definiert werden, da sonst die folgenden Arrays ebenfalls nicht ausgegeben werden.

FIND Ausdruck
sucht einen beliebigen Ausdruck in einem Programm und gibt die entsprechende Programmzeile aus. Will man einen String suchen, ist es zweckmäßig, auf die Anführungszeichen am Ende zu verzichten. Zu suchende Basic-Befehle dürfen nicht abgekürzt werden.

Änderung der Standard-Werte:
In Tabelle 2 finden Sie noch einige wichtige Adressen zur Anpassung des Programms an individuelle Bedürfnisse. So ist es durchaus möglich, Paradoxon-Basic zusammen mit dem 64'er DOS, das in Ausgabe 3/86 und 4/86 veröffentlicht wurde, zu betreiben. Neue Befehle und eine schnelle Floppystation bilden so eine gutes Gespann.

(Robert Bartz/Stefan Willmeroth)

Die Macht der Rekursion

Manche Programmprojekte lassen sich mit dem Basic 2.0 des C64 nur schwer durchführen. »Rekursiv-Basic« heißt das Zauberwort, das bis zu 2454 Verschachtelungen von sich selbst aufrufenden Prozeduren erlaubt. Auch strukturiertes Programmieren wird auf diese Weise unterstützt.

Um mit »Rekursiv-Basic« zu arbeiten, müssen Sie Listing 1 mit dem MSE abtippen und speichern. Das nur sieben Blöcke lange Programm wird mit RUN gestartet. Die weiteren Listings sind Demonstrations- und Hilfsprogramme, die Ihnen die erstaunlichen Fähigkeiten dieser Basic-Erweiterung aufzeigen. Sie werden später noch genauer erläutert.

Mit Rekursiv-Basic sind auch eigene Basic-Erweiterungen in Form von namentlich aufrufbaren Prozeduren (Unterprogramme) ohne weiteres in Basic realisierbar. Durch die Definition von lokalen Variablen, die nur in dem Unterprogramm gültig sind, stören sich das Hauptprogramm und die Prozeduren nicht. Die Variablen werden unter dem Kernel-ROM zwischengespeichert und verbrauchen somit keinen Basic-Speicher. Um auch umfangreiche Rekursionen (eine rekursive Prozedur ist eine Prozedur, die sich selbst aufruft) durchführen zu können, wurde der Stack, wo die Rücksprungadressen der Prozeduren abgelegt werden, stark vergößert, so daß jetzt maximal 2454 Prozedur-Verschachtelungen auftreten können. Die Vergrößerung des Stack führt auch dazu, daß bis zu 681 FOR-NEXT-Schleifen ineinander verschachtelt werden können. Auch der Stack verbraucht keinen Basic-Speicherplatz.

Gut strukturierte Programme

Nach dem Start von Rekursiv-Basic mit RUN stehen die neuen Befehle zur Verfügung. Sie sind in Tabelle 1 aufgeführt. Falls Sie mit den veschiedenen Klammern in den Befehlsbezeichnungen Probleme haben sollten, finden Sie in Tabelle 2 eine kurze Erläuterung dieser Zeichen.

£PROC »Name« [»Parameterliste«]

Der PROC-Befehl deklariert eine Prozedur mit dem Namen »Name«, dem optional eine Parameterliste folgen kann. Zwischen PROC und dem Namen muß ein Leerzeichen gesetzt werden; falls eine Parameterliste folgt, muß diese ebenfalls durch ein Leerzeichen vom Namen getrennt werden. Einzelne Parameter werden durch Kommata getrennt.

Wichtig: PROC darf nur am Anfang einer Zeile stehen und nie direkt angesprungen oder ausgeführt werden, das heißt, daß Prozeduren nur durch den Prozeduraufruf-Befehl (siehe unter 4.) ausgeführt werden können. Ein Beispiel dazu:

```
10 £PROC Test
20 £END
```

Nach der Eingabe von RUN käme sofort die Fehlermeldung »CAN'T EXECUTE PROC«. Richtig muß es lauten:

```
10 GOTO 100
20 £PROC Test
30 £END
100 REM Hier kann das Programm folgen
```

Übergabe von Parametern an Prozeduren

Die Paramterliste besteht aus einer Auflistung von Variablen, die vom Hauptprogramm oder einer übergeordneten Prozedur übergeben werden sollen. Die Variablen werden, wie in allen entsprechenden Befehlen von Rekursiv-Basic durch Kommata getrennt. Dabei kann die Tabelle sich aus allen Variablentypen (Fließkomma, Integer, String) zusammensetzen. Beispiel:

```
1000 £PROC Test A,B%,C$
1010 £END
```

Bei dem Aufruf dieser Prozedur werden die Variablen A, B% und C$ (sofern sie schon existieren) auf einen speziellen Stack (Heap) gelegt und erhalten dann anschließend die Werte, die in der Parameterliste des Aufrufs stehen. Wichtig ist hierbei, daß die Variablentypen der Parameter im Prozedur-Aufruf mit den Variablentypen in der Parameterliste des PROC-Befehls übereinstimmen. Beispiel:

```
5 A=1.2 :C$= "C-64"
10 GOTO 100
20 £PROC Test A, B%, C$
30 PRINT A; B%; C$
40 £END
100 !Test 3.14, 10, "Hallo"
110 END
```

Befehl	Erläuterung
£PROC »Name« [»Parameterliste«]	Prozedur-Deklaration
£VAR »Parameterliste«	Lokale Variablen definieren
£END	Prozedur-Ende
!»Name« [»Parameter«]	Prozedur-Aufruf

Tabelle 1. Die Prozedur-Befehle von Rekursiv-Basic

Parameter	Bedeutung
»Name«	Steht für einen beliebigen Namen, der bis auf Leerzeichen alle Zeichen enthalten darf.
»Parameterliste«	Eine Aufzählung von Variablen, die mit einem Komma voneinander getrennt werden müssen.
[...]	Alle Ausdrücke, die in eckigen Klammern ([...]) stehen, sind optional. Der Ausdruck muß nicht unbedingt folgen, dies hängt von der Art der Prozedur ab.

Tabelle 2. Erklärung der in Tabelle 1 verwendeten Parameter zur Handhabung von Prozeduren

Steckbrief Rekursiv-Basic

- Prozedur-Aufrufe mit Parameterübergabe
- Lokale und globale Variablen
- Große Verschachtelungstiefen und Rekursion werden möglich
- Prozeduren können als Funktionsersatz dienen (Fakultät etc.)
- Prozeduren sind auch im Direktmodus ausführbar
- Prozedur-Aufrufe sind an jeder Stelle im Programm möglich. Jede Prozedur kann jede andere oder auch sich selbst aufrufen
- Pascal-ähnliche Strukturen werden möglich
- Pseudo-Stack (Heap) mit 12 oder wahlweise 24 KByte
- Weder Stack noch Rekursiv-Basic belegen den Basic-Speicher
- Bis zu 2454 Prozedur-Hierarchien (Verschachtelungen) bei 12 KByte Stack
- Bis zu 681 Verschachtelungen bei FOR-NEXT-Schleifen (12 KByte Stack)

Anwendungen

- Strukturiertes Programmieren
- Eigene Befehlserweiterungen
- Programme mit künstlicher Intelligenz
- Schnelle Sortierverfahren (Quicksort etc.)

Nach der Prozedur-Ausführung werden die ursprünglichen Variablen wieder hergestellt (im Beispiel die in Zeile 5 definierten Variablen A und C$). Rekursiv-Basic kehrt danach ins Hauptprogramm oder in die Prozedur zurück, welche die gerade ausgeführte aufgerufen hat. Diese Art von »Übergabevariablen« (im Beispiel: A, B%, C$) nennt man Werteparameter, das heißt, daß diese Variablen am Anfang einer Prozedur Werte erhalten, diese aber nach der Ausführung nicht an das Hauptprogramm oder die übergeordnete Prozedur zurückgeben. Die in der Prozedur benutzten Variablen werden wieder »vergessen«. Dazu ein Beispiel:

```
10 A=15: B=10
20 !Test A
30 PRINT A, B
40 END
100 £PROC Test B
110 B= B+5: PRINT B
120 £END
```

Dieses Programm liefert die Bildschirmausgabe 20, 15 und 10. Es funktioniert folgendermaßen: Zuerst werden den Variablen A und B die Werte 15 und 10 zugewiesen, dann wird die Prozedur »Test« aufgerufen, wobei in der Prozedur eine neue Variable B angelegt und dieser der Wert von A zugewiesen wird. Nachdem die Prozedur abgearbeitet ist, wird die neue Variable B vergessen, die alte Variable B wird wieder vom Heap (erweiterten Stack) geholt und das Hauptprogramm fährt an der Stelle nach dem Prozedur-Aufruf fort.

Variablenübergabe an Prozeduren

Bisher können Sie allerdings nur Variablen an die Prozeduren übergeben und nicht umgekehrt. Dieses Problem wird durch die sogenannten Variablenparameter gelöst. Sie ermöglichen nämlich die Rückgabe von Werten oder Ergebnissen aus der ausgeführten Prozedur an das Hauptprogramm oder die übergeordnete Prozedur. Eine Variable in der Parameterliste der Prozedur-Deklaration wird als Variablenparameter gekennzeichnet, indem ihr ein Doppelkreuz (»#«) vorangestellt wird. Der Wert dieser Variablen wird nach der Ausführung der Prozedur nicht wieder vergessen, sondern der Variablen in der Parameterliste des Aufrufs zugewiesen. Ein Beispiel zu den Variablenparametern:

```
10 !Lies A
20 PRINT A
30 END
100 £PROC Lies #B
110 INPUT B
120 £END
```

Die Prozedur »Lies« hat die gleiche Funktion wie der Basic-Befehl INPUT »numerische Variable«.

£VAR »Parameterliste«

Der VAR-Befehl wird in Prozeduren dazu verwendet, lokale Variablen zu schaffen. Alle in einer Prozedur benutzten Variablen sollten hier aufgeführt werden, damit nicht irgendwelche Variablen des Hauptprogramms oder einer übergeordneten Prozedur verändert werden. Sollten nicht alle Variablen hinter einen VAR-Befehl passen, so ist es auch möglich, in der Folgezeile einen weiteren VAR-Befehl anzubringen.

Wichtig: Feld-Elemente oder gar ganze Felder können nicht als lokale Variablen deklariert werden. Wenn Sie in Prozeduren also Felder benötigen, können Sie diese zwar mit dem DIM-Befehl schaffen, doch bestehen sie auch nach der Rückkehr aus dieser Prozedur immer noch, das heißt, Sie können dann nicht noch einmal ein Feld gleichen Typs und Namens in irgendeiner anderen Prozedur deklarieren. Ein Beispiel zum VAR-Befehl:

```
10 A = 100: B$ = "TEST"
20 !Test
30 PRINT A, B$
40 END
100 £PROC Test
110 £VAR A, B$
120 A = 33: B$ = "64'ER"
130 PRINT A,B$
140 £END
```

Dieses kurze Programm liefert die Bildschirmausgaben »33«, »64'ER«, »100« und »Test«.

PROC NOT DECLARED	Es wurde eine nicht vorhandene Prozedur aufgerufen
CAN'T EXECUTE PROC	Es wurde versucht, einen PROC-Befehl auszuführen
STACK OVERFLOW	Es wurden entweder zu viele Prozeduren verschachtelt oder zu viele lokale Variablen geschaffen
STACK EMPTY	Der END-Befehl wurde ohne vorhergehenden Prozedur-Aufruf ausgeführt

Tabelle 3. Die Fehlermeldungen von Rekursiv-Basic

£END

Der Befehl END wird dazu verwendet, eine Prozedur-Ausführung zu beenden. Sämtliche Variablen, die mit dem VAR-Befehl geschaffen wurden sowie alle Werteparameter werden wieder »vergessen«. Bei Variablenparametern erfolgt nun die Zuweisung der jeweiligen Werte an die Aufrufvariablen. Anschließend wird das Programm hinter dem Prozedurenaufruf fortgeführt.

!»Name« [»Parameter 1«], [»Parameter 2«]

Dieser Befehl führt die Prozedur mit dem Namen »Name« aus, die an einer beliebigen Stelle im Programm stehen kann. Wichtig ist hierbei, daß die aufgeführten Parameter jeweils gleichen Typs wie die Variablen im Prozedur-Kopf (hinter dem PROC-Befehl) sind. Wenn Variablenparameter verwendet werden, ist darauf zu achten, daß an der entsprechenden Position im Aufruf ebenfalls eine Variable steht und nicht eine Zahl oder ein String-Ausdruck, damit der richtige Wert der Prozedur-Variablen nach der Ausführung zugewiesen werden kann. Die Funktionsweise des »!«-Befehles läßt sich auch an dem vorangegangenen Beispiel ersehen.

Rekursiv-Basic stellt eine Reihe eigener Fehlermeldungen bereit. Diese finden Sie in Tabelle 3. Zusätzlich sollten Sie einige Punkte beachten:

1. Nach einem THEN-Befehl sollte grundsätzlich ein Doppelpunkt »:« folgen, da es sonst eventuell bei manchen Basic-Befehlen zu verschiedenen Fehlermeldungen kommen kann. Der Grund hierfür ist der »verbogene« Vektor des Betriebssystems, der für die Abarbeitung der Basic-Befehle zuständig ist.

2. Bei der Verwendung von Variablenparametern ist darauf zu achten, daß der Variablenname im Prozedur-Aufruf nicht gleich dem Variablennamen im Prozedur-Kopf ist. Wählen Sie aus diesem Grund für Variabenparameter in Prozeduren möglichst seltene Namen, wie z.B. ZQ$ oder QJ%, um nicht bei jedem Prozedur-Aufruf einen Blick auf die Parameterliste der Prozedur werfen zu müssen.

3. Da die Parameter bei einem Prozedur-Aufruf schrittweise nacheinander übergeben werden, ist darauf zu achten, daß eine Variable, die im Prozedur-Kopf am Anfang sind, im Aufruf an der entsprechenden Stelle steht.

Zum besseren Verständnis der Punkte 2. und 3. sehen Sie sich bitte das folgende Beispiel an:

```
5   A = 10: B = 17
10  !Test A, B
20  END
100 £PROC Test B, A
10  PRINT B, A
120 £END
```

Die Bildschirmausgabe ist »10, 10«, da beim Einsprung in die Prozedur (Zeile 100) zuerst die Variable B gesichert und ihr der Wert von A zugewiesen wird. Anschließend wird die Variable A gesichert und ihr der Wert von B zugewiesen, der aber mittlerweile schon geändert worden ist. Um dieses Problem nun zu umgehen, sollte man einfach eine Hilfsvariable benutzen:

```
5   A = 10: B = 17
10  C = B: !Test A, C
20  END
100 £PROC Test B,A
110 PRINT B,A
120 £END
```

Bildschirmausgabe: 10, 17.

Die Speicherbelegung von Rekursiv-Basic

Rekursiv-Basic belegt den Speicher von \$C000 is \$C7FF mit der eigentlichen Basic-Erweiterung und das RAM unter dem I/O-Bereich und dem Kernal (\$D000 bis \$FFFF) mit Daten und dem Pseudo-Stack. Folgende Speicherstellen der Zeropage werden verwendet:

\$F9/FA	Stackpointer
\$FB/FC	Hilfszeiger
\$02	Hilfsregister

Ferner wurden noch die ersten acht Byte des Kassettenpuffers für Hilfszwecke gebraucht (\$033C bis \$0343). Das Programm »Türme von Hanoi« (Listing 2) löst nach der Eingabe der Scheibenanzahl das bekannte Türme-von-Hanoi-Problem (Bild 1), bei dem es darum geht, einen aus verschieden großen Scheiben bestehenden Turm abzubauen und diesen auf einem anderen Platz wieder zusammenzusetzen. Dabei müssen jedoch die folgenden Regeln beachtet werden:

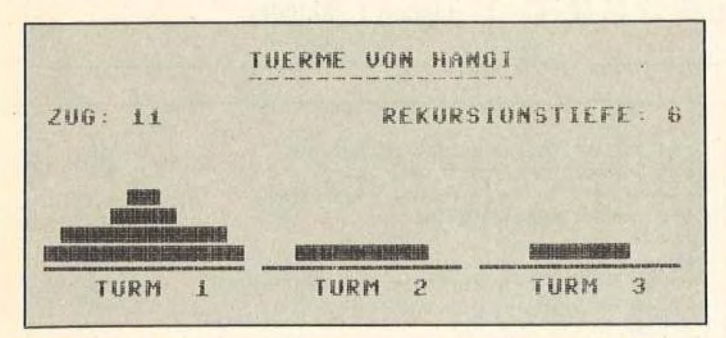


Bild 1. Das Programm »Türme von Hanoi« bei der Arbeit

Befehl	Bedeutung
CLS	Bildschirm löschen
COLOR Rahmenfarbe, Hintergrundfarbe	Bildschirmfarben setzen
CLREOL	Rest der aktuellen Bildschirmzeile löschen
CURSOR X, Y	Cursor an die Position X, Y setzen
PRTAT X, Y, A\$	Den Text-String A\$ an der Position X, Y ausgeben. Anstelle A\$ kann auch Text in Anführungszeichen eingegeben werden
LINIE I	Eine horizontale Linie (CHR\$(192) mit der Länge »I« ab der momentanen Cursorposition zeichnen
WAITKEY A\$, B\$, Modus	Ein Zeichen, das in B\$ vorkommt, von Tastatur als A\$ einlesen. Sollen alle Charakter einlesbar sein, so muß B\$ leer (" ") sein. Ist Modus ungleich 0, so wird der Cursor nicht blinkend eingeschaltet.
POSITION Such\$, Quell\$, POS	Suche nach einem String: POS enthält die Position ab der Such\$ in Quell\$ beginnt. Ist Such\$ nicht in Quell\$ vorhanden, ist POS gleich 0.
CENTER Text\$	Text (Text\$) zentrieren
FETCH A\$, I	String A\$ mit der maximalen Länge »I« von der Tastatur einlesen
LPRINT Sekundäradr., Text\$	Den String Text\$ mit gewünschter Sekundäradresse auf den Drucker ausgeben
STATUS Er, Er\$, Tr, Se	Holt den Floppystatus in die Variablen Er, Er\$, Tr, Se. Nur im Programmodus ausführbar.
BEEP H, L	Spielt einen Ton der Höhe H mit der Länge L. Für H können Werte bis 65535 eingegeben werden.
CATALOG	Directory ausgeben

Tabelle 4. Die Befehle der »Library«

1. Es darf niemals eine größere Scheibe auf eine kleinere gelegt werden.
2. Man hat nur drei Ablageplätze für die Scheiben.

Die Hauptprozedur »Hanoi« ist rekursiv, die Rekursionstiefe wird während des Programmablaufs auf dem Bildschirm angezeigt. Bitte beachten Sie bei der Eingabe des Programms, daß die Befehle CLRSCR, COLOR, WAITKEY und CURSOR zusammengeschrieben werden müssen, da sonst ein SYNTAX ERROR auftritt.

Eine Basic-Erweiterung: Die Library

Die Library (Listing 3) stellt eine Basic-Erweiterung dar, die Sie an Ihre selbstgeschriebenen Programme anhängen können, sofern die Zeilennummern ab 50000 unbenutzt sind. Eine Übersicht der neuen Befehle finden Sie in Tabelle 4.

Wenn Sie diese neuen Befehle benutzen wollen, können Sie die Library mit dem Programm »MERGER« (Listing 4) an Ihr Programm anhängen. Dazu muß die letzte Zeilennummer Ihres Programms allerdings kleiner als 50000 sein. »MERGER« ist ein kleines Maschinenprogramm, das mit dem MSE abgetippt wird. Es ist einfach zu bedienen:

Nach dem Laden von »MERGER« können Sie Ihr Hauptprogramm laden und dann mit

```
SYS 704,"Name",Geräteadresse
```

das gewünschte Programm anhängen.

(Thomas Kolbe/Andreas Lietz)

```
Name : rekursiv-basic     0801 0ee4
-----------------------------------
0801 : 1b 08 c2 07 9e 32 30 38  5f
0809 : 30 20 42 59 20 54 48 4f  69
0811 : 4d 41 53 20 4b 4f 4c 42  bd
0819 : 45 00 00 00 ff ff ff a9  b2
0821 : 4a a0 08 85 fb 84 fc a9  99
0829 : c0 85 fe a0 00 84 fd b1  ff
0831 : fb 91 fd c8 d0 f9 e6 fc  00
0839 : e6 fe a5 fe c9 c7 d0 ef  e5
0841 : 20 00 c0 70 44 a6 4c ae  9d
0849 : a7 ad 09 03 c9 c0 f0 1d  0a
0851 : a9 45 a0 c6 20 1e ab ad  9b
0859 : 08 03 8d 2d c0 ad 09 03  8f
0861 : 8d 2e c0 a9 2f a0 c0 8d  81
0869 : 08 03 8c 09 03 a9 00 a0  f6
0871 : d0 85 f9 84 fa 60 c3 e4  9e
0879 : 20 73 00 f0 1d c9 21 f0  f7
0881 : 55 c9 5c f0 22 c9 81 d0  08
0889 : 03 4c 46 c2 c9 82 d0 03  96
0891 : 4c b5 c2 c9 8a d0 03 4c  75
0899 : a3 c3 c6 7a a5 7a c9 ff  74
08a1 : d0 02 c6 7b 6c 2d c0 20  07
08a9 : 73 00 c9 80 d0 03 4c 72  da
08b1 : c1 c9 56 d0 17 20 73 00  47
08b9 : c9 41 d0 10 20 73 00 c9  8a
08c1 : 52 d0 09 20 73 00 20 d8  2b
08c9 : c4 4c ae a7 a9 f7 a0 c5  bc
08d1 : 20 1e ab 4c 62 a4 a5 2b  ad
08d9 : a4 2c 8d fc 03 8c fd 03  29
08e1 : 20 05 c4 b0 0a a9 09 a0  1e
08e9 : c6 20 1e ab 4c 62 a4 e6  f5
08f1 : fb d0 02 e6 fc 20 30 c4  cd
08f9 : 90 e6 a0 01 b1 fb c9 20  a7
0901 : f0 10 c9 3a f0 0c c9 00  4a
0909 : f0 08 d1 7a d0 d2 c8 4c  20
0911 : b3 c0 20 73 00 e6 fb d0  64
0919 : 02 e6 fc 88 d0 f4 20 ae  71
0921 : c4 a5 7a 8d 3c 03 a5 7b  72
```

Listing 1. »Rekursiv-Basic«, bitte mit dem MSE (Seite 159) eingeben

```
0929 : 8d 3d 03 a5 fb 85 7a a5  eb
0931 : fc 85 7b 20 d8 c4 a5 fb  15
0939 : 85 7a a5 fc 85 7b 20 79  ac
0941 : 00 d0 03 4c 5d c1 c9 23  45
0949 : d0 03 20 af c3 20 8b b0  66
0951 : 85 49 84 4a 20 79 00 f0  95
0959 : 03 20 fd ae a5 7a 85 fb  fe
0961 : a5 7b 85 fc ad 3c 03 85  99
0969 : 7a ad 3d 03 85 7b a5 0e  50
0971 : 48 a5 0d 48 20 9e ad 68  57
0979 : 2a 20 90 ad d0 0f 68 10  d4
0981 : 06 20 c4 a9 4c 48 c1 20  4c
0989 : d6 a9 4c 48 c1 68 20 2c  88
0991 : aa 20 79 00 f0 03 20 fd  4d
0999 : ae a5 7a 8d 3c 03 a5 7b  d4
09a1 : 8d 3d 03 4c ed c0 a5 fb  8b
09a9 : 85 7a a5 fc 85 7b ad fe  5d
09b1 : 03 85 39 ad ff 03 85 3a  1d
09b9 : 4c ae a7 20 84 c4 8d 3e  6b
09c1 : 03 29 80 f0 03 4c 1a c2  17
09c9 : ad 3e 03 c9 01 f0 6f c9  78
09d1 : 05 d0 03 4c 0f c2 8d 3e  42
09d9 : 03 20 84 c4 85 4a 20 84  da
09e1 : c4 85 49 ad 3e 03 c9 04  9b
09e9 : f0 14 c9 03 f0 0c a0 06  b4
09f1 : 20 84 c4 91 49 88 10 f8  c2
09f9 : 30 c1 a0 03 d0 f2 20 84  c0
0a01 : c4 aa f0 0c ca 20 84 c4  21
0a09 : 9d 00 c7 ca e0 ff d0 f5  2f
0a11 : 20 84 c4 aa a9 00 9d 00  0b
0a19 : c7 a0 c7 20 87 b4 a0 01  c9
0a21 : 20 84 c4 91 49 88 20 84  49
0a29 : c4 91 49 a5 49 18 69 02  bc
0a31 : 85 49 a5 4a 69 00 85 4a  4f
0a39 : 20 2c aa 4c 72 c1 20 84  62
0a41 : c4 85 39 20 84 c4 85 3a  13
0a49 : 20 84 c4 85 7a 20 84 c4  d1
0a51 : 85 7b 20 f8 a8 4c ae a7  b2
0a59 : a2 11 20 84 c4 ca d0 fa  f8
0a61 : 4c 72 c1 20 84 c4 85 bc  59
0a69 : 20 84 c4 85 bb 20 84 c4  05
0a71 : 85 23 20 84 c4 85 22 ad  7d
0a79 : 3e 03 38 e9 82 aa bd 43  7f
0a81 : c2 a8 b1 22 91 bb 88 10  81
0a89 : f9 4c 72 c1 04 01 02 a9  21
0a91 : 80 85 10 20 73 00 20 a5  df
0a99 : a9 20 79 c5 20 06 a9 18  72
0aa1 : 98 65 7a 08 20 56 c4 28  a4
0aa9 : a5 7b 69 00 20 56 c4 a5  79
0ab1 : 3a 20 56 c4 a5 39 20 56  7b
0ab9 : c4 a9 a4 20 ff ae 20 8d  90
0ac1 : ad 20 8a ad a5 66 09 7f  87
0ac9 : 25 62 85 62 20 5c c5 a9  1c
0ad1 : bc a0 b9 20 a2 bb 20 79  cb
0ad9 : 00 c9 a9 d0 06 20 73 00  71
0ae1 : 20 8a ad 20 2b bc 20 56  7b
0ae9 : c4 20 5c c5 a5 4a 20 56  67
0af1 : c4 a5 49 20 56 c4 a9 05  1a
0af9 : 20 56 c4 4c ae a7 20 73  8e
0b01 : 00 20 c8 c4 90 05 a2 0a  ac
0b09 : 4c 37 a4 20 84 c4 c9 05  be
0b11 : d0 f4 20 79 00 d0 0f 20  96
0b19 : 84 c4 85 49 20 84 c4 85  ce
0b21 : 4a a4 49 4c f9 c2 20 8b  e6
0b29 : b0 85 49 84 4a 20 84 c4  c0
0b31 : a8 20 84 c4 c5 4a f0 03  1b
0b39 : 4c 9b c3 c4 49 f0 03 4c  9d
0b41 : 9b c3 8d 41 03 8c 40 03  e5
0b49 : a2 05 20 af c5 20 c0 c5  58
0b51 : 20 d4 c5 a2 06 20 af c5  4d
0b59 : a0 00 b1 22 85 66 a5 49  5f
0b61 : a4 4a 20 ca c5 20 67 b8  f8
0b69 : 20 d0 bb a2 0b 20 af c5  30
0b71 : a0 04 20 c0 c5 a2 00 b1  08
0b79 : 22 9d 00 c7 e8 88 10 f7  66
0b81 : 20 ca c5 a9 00 a0 c7 20  11
0b89 : 5b bc 8d 42 03 a2 06 20  8c
0b91 : af c5 20 c0 c5 a0 00 ad  00
0b99 : 42 03 d1 22 f0 31 a2 0f  57
0ba1 : 20 af c5 a0 00 b1 22 85  3f
0ba9 : 7a c8 b1 22 85 7b c8 b1  f3
0bb1 : 22 85 3a c8 b1 22 85 39  f2
0bb9 : 20 ca c5 ad 41 03 20 56  bf
0bc1 : c4 ad 40 03 20 56 c4 a9  e7
0bc9 : 05 20 56 c4 4c ae a7 20  26
0bd1 : ca c5 a2 0f 20 a8 c5 20  a7
0bd9 : 79 00 c9 2c d0 ee 20 73  36
0be1 : 00 4c b5 c2 a2 0f 20 a8  41
0be9 : c5 4c c2 c2 20 24 c0 20  44
0bf1 : 73 00 20 71 a8 4c ae a7  92
0bf9 : 20 73 00 a5 7a 8d 40 03  a3
0c01 : a5 7b 8d 41 03 20 8b b0  b0
0c09 : 20 56 c4 98 20 56 c4 a5  ac
0c11 : 0e 8d 42 03 a5 0d 8d 43  56
0c19 : 03 ad 3c 03 85 7a ad 3d  c0
0c21 : 03 85 7b 20 8b b0 20 56  35
0c29 : c4 98 20 56 c4 ad 40 03  cd
0c31 : 85 7a ad 41 03 85 7b ad  2c
0c39 : 43 03 d0 0e ad 42 03 d0  8e
0c41 : 05 a9 82 4c 56 c4 a9 83  7e
0c49 : d0 f9 a9 84 d0 f5 ad fc  7e
0c51 : 03 ac fd 03 d0 02 18 60  c8
0c59 : 85 fb 84 fc a0 00 b1 fb  65
0c61 : 99 fc 03 c8 c0 04 d0 f6  b0
0c69 : b1 fb c9 5c d0 e0 e6 fb  bd
0c71 : d0 02 e6 fc 88 d0 f7 38  fb
0c79 : 60 a9 1d a0 c6 4c 37 c4  3e
0c81 : 85 bb 84 bc a0 00 b1 bb  e5
0c89 : f0 0a d1 fb f0 02 18 60  b2
0c91 : c8 4c 3d c4 e6 fb d0 02  fd
0c99 : e6 fc 88 d0 f7 38 60 48  8d
0ca1 : a5 f9 c9 f0 d0 11 a5 fa  f5
0ca9 : c9 ff d0 0b 68 a9 26 a0  b5
0cb1 : c6 20 1e ab 4c 62 a4 68  c0
0cb9 : 20 c0 c5 84 02 a0 00 91  84
0cc1 : f9 e6 f9 d0 02 e6 fa 20  49
0cc9 : ca c5 a4 02 60 20 c8 c4  93
0cd1 : 90 0a a9 37 a0 c6 20 1e  b5
0cd9 : ab 4c 62 a4 20 c0 c5 84  00
0ce1 : 02 c6 f9 a5 f9 c9 ff d0  09
0ce9 : 02 c6 fa a0 00 b1 f9 48  27
0cf1 : a4 02 20 ca c5 68 60 a5  64
0cf9 : 7b 20 56 c4 a5 7a 20 56  0e
0d01 : c4 a5 3a 20 56 c4 a5 39  bf
0d09 : 20 56 c4 a9 01 20 56 c4  af
0d11 : 60 a5 f9 c9 00 d0 08 a5  ed
0d19 : fa c9 d0 d0 02 38 60 18  da
0d21 : 60 20 79 00 d0 01 60 c9  1a
0d29 : 23 d0 03 20 73 00 20 8b  48
0d31 : b0 38 e9 02 b0 01 88 85  f8
0d39 : bb 84 bc a0 00 b1 bb 20  36
0d41 : 56 c4 c8 c0 03 d0 f6 8d  f1
0d49 : 3e 03 a5 0d d0 32 a5 0e  65
0d51 : d0 25 b1 bb 20 56 c4 c8  f1
0d59 : c0 07 d0 f6 a2 02 a5 bb  f8
0d61 : 20 56 c4 a5 bc 20 56 c4  42
0d69 : 8a 20 56 c4 20 79 00 f0  e1
0d71 : b5 20 fd ae 4c de c4 b1  be
0d79 : bb 20 56 c4 a2 03 d0 de  b6
0d81 : b1 bb 85 49 c8 b1 bb 85  ae
0d89 : 4a a0 00 ad 3e 03 f0 0b  af
0d91 : b1 49 20 56 c4 c8 cc 3e  fc
0d99 : 03 d0 f5 ad 3e 03 20 56  61
0da1 : c4 a2 04 d0 b9 20 1b bc  54
0da9 : a5 65 20 56 c4 a5 64 20  1f
0db1 : 56 c4 a5 63 20 56 c4 a5  52
0db9 : 62 20 56 c4 a5 61 20 56  ec
0dc1 : c4 60 20 c8 c4 b0 30 20  a9
0dc9 : 84 c4 c9 05 d0 20 a2 0f  79
0dd1 : 20 84 c4 a8 20 84 c4 c4  3c
0dd9 : 49 d0 07 c5 4a d0 03 4c  d5
0de1 : a8 c5 20 56 c4 98 20 56  7d
0de9 : c4 a9 05 4c 56 c4 4c 56  b6
0df1 : c4 20 84 c4 ca d0 fa 60  5f
0df9 : 86 bb a5 f9 38 e5 bb a4  f0
0e01 : fa b0 01 88 85 22 84 23  66
0e09 : 60 48 78 a5 01 29 f8 85  a8
0e11 : 01 68 60 48 a5 01 09 07  fc
0e19 : 85 01 68 58 60 a0 00 b1  b2
0e21 : 22 85 65 c8 b1 22 85 64  83
0e29 : c8 b1 22 85 63 c8 b1 22  8b
0e31 : 85 66 09 80 85 62 c8 b1  2e
0e39 : 22 85 61 a0 00 84 70 60  31
0e41 : 11 3f 43 41 4e 27 54 20  9b
0e49 : 45 58 45 43 20 50 52 4f  e1
0e51 : 43 00 11 3f 50 52 4f 43  1c
0e59 : 20 4e 4f 54 20 44 45 43  be
0e61 : 4c 41 52 45 44 00 50 52  b5
0e69 : 4f 43 00 56 41 52 00 11  ee
0e71 : 3f 53 54 41 43 4b 20 4f  45
0e79 : 56 45 52 46 4c 4f 57 00  6c
0e81 : 11 3f 53 54 41 43 4b 20  2d
0e89 : 45 4d 50 54 59 00 93 11  19
0e91 : 20 2a 2a 2a 2a 2a 2a 2a  87
0e99 : 20 20 20 20 20 52 45 4b  16
0ea1 : 55 52 53 49 56 2d 42 41  78
0ea9 : 53 49 43 20 20 20 20 20  3a
0eb1 : 2a 2a 2a 2a 2a 2a 2a 0d  77
0eb9 : 0d 20 47 45 53 43 48 52  66
0ec1 : 49 45 42 45 4e 20 56 4f  c4
0ec9 : 4e 20 20 54 48 4f 4d 41  71
0ed1 : 53 20 4b 4f 4c 42 45 20  1d
0ed9 : 20 49 4e 20 31 39 38 36  60
0ee1 : 0d 00 00 a0 b5 a9 1a 20  54
```

Listing 1. (Schluß)

```
0 REM      >>> TUERME VON HANOI <<<                 <162>
1 REM                                               <063>
2 REM    DEMO ZUM REKURSIV-BASIC VON:               <134>
3 REM                                               <065>
4 REM      ***** THOMAS KOLBE *****                 <087>
5 REM                                               <067>
6 REM          DEMO PROGRAMM VON:                   <011>
7 REM                                               <069>
8 REM        *** ZUHEIR URWANI ***                  <214>
9 REM                                               <071>
10 :                                                <242>
20 !CLR SCR:!COL OR 0                               <033>
30 !CURS OR 12,1:PRINT"TUERME VON HANOI"            <220>
40 !CURS OR 12,2:PRINT"----------------"            <206>
50 !EINGABE:FOR I=0 TO 2                            <232>
60 !CURS OR I*13,22:PRINT "DDDDDDDDDDDD"            <212>
70 !CURS OR I*13+3,23:PRINT "TURM ";I+1;            <181>
80 NEXT I:PRINT "{CYAN}";                           <105>
100 TU (1,0)=AN                                     <047>
110 TU (2,0)=0                                      <162>
120 TU (3,0)=0                                      <204>
125 FOR I=1 TO AN                                   <053>
130 TU (1,I)=(AN+1-I)*2                             <068>
140 !SCHEIBE 0,22-I,(AN+1-I)*2:NEXT I               <125>
145 ZUG=0:!CURS OR 0,4:PRINT "ZUG:"ZUG              <083>
147 REK=1:!CURS OR 20,4                             <138>
149 PRINT "REKURSIONSTIEFE:"REK                     <012>
150 !HANOI AN,1,2,3,REK                              <177>
160 !CURS OR 16,6:PRINT "FERTIG !"                   <044>
170 !WAIT KEY:GOTO 20                               <062>
180 :                                               <156>
190 :                                               <166>
10000 REM    PROZEDUREN ZUM PROGRAMM                <160>
10010 :                                             <080>
10020 £PROC CLR SCR                                 <179>
10030 PRINT "{CLR,LIG.BLUE}"                        <007>
10040 £END                                          <251>
10050 :                                             <120>
10060 :                                             <130>
10070 £PROC COL OR FARBE                            <129>
10080 POKE 53280,FARBE:POKE 53281,FARBE             <160>
10090 £END                                          <045>
10100 :                                             <170>
10110 :                                             <180>
10120 £PROC SCHEIBE X,Y,N                           <100>
10130 £VAR J                                        <243>
10140 !CURS OR X,Y                                  <104>
10145 IF N=12 THEN 10160                            <072>
10150 FOR J=1 TO 6-N/2:PRINT " ";:NEXT              <151>
10160 FOR J=1 TO N:PRINT "{RVSON}T{RVOFF}"
      ;:NEXT                                        <108>
10170 £END                                          <125>
```

```
10180 :                                  <250>
10190 :                                  <004>
10200 £PROC HANOI N,A,B,C,R              <123>
10205 £VAR D,E,F                         <137>
10207 !CURS OR 36,4:PRINT R              <240>
10210 IF N<1 THEN:£END                   <148>
10215 D=A:E=B:F=C                        <120>
10220 !HANOI N-1,D,F,E,R+1               <220>
10240 !VERSETZE A,B                      <203>
10250 !HANOI N-1,F,E,D,R+1               <219>
10255 !CURS OR 36,4:PRINT R              <034>
10260 £END                               <217>
10270 :                                  <086>
10320 :                                  <136>
10330 £PROC WAIT KEY:£VAR A$             <096>
10340 GET A$:IF A$="" THEN 10340         <074>
10350 £END                               <051>
10360 :                                  <176>
10370 :                                  <186>
10380 £PROC VERSETZE VO,NA               <046>
10390 £VAR V,N,B                         <193>
10395 ZUG=ZUG+1:!CURS OR 4,4:PRINT ZUG   <022>
10400 V=TU (VO,0)                        <178>
10410 N=TU (NA,0)+1                      <090>
10420 B=TU (VO,V)                        <015>
10430 TU (NA,N)=B                        <054>
10440 !LOESCHE (VO-1)*13,22-V            <053>
10450 !SCHEIBE (NA-1)*13,22-N,B          <030>
10460 TU (VO,0)=TU (VO,0)-1              <128>
10470 TU (NA,0)=TU (NA,0)+1              <245>
10480 £END                               <181>
10490 :                                  <050>
10500 :                                  <062>
10510 £PROC LOESCHE X,Y                  <072>
10520 !CURS OR X,Y:PRINT "{12SPACE}"     <030>
10530 £END                               <233>
10540 :                                  <102>
10550 :                                  <112>
10560 £PROC CURS OR X,Y                  <047>
10570 POKE 781,Y:POKE 782,X              <215>
10580 POKE 783,PEEK(783)AND 254          <002>
10590 SYS 65520                          <120>
10600 £END                               <047>
10610 :                                  <172>
10620 :                                  <182>
10630 £PROC EINGABE:£VAR A$              <076>
10635 A$="{28SPACE}"                     <250>
10700 !CURS OR 0,4                       <223>
10710 PRINT"ANZAHL DER SCHEIBEN (1-6)?"  <186>
10720 !CURS OR 9,6:INPUT AN              <232>
10730 IF AN>0 AND AN<7 THEN 10750        <177>
10740 !CURS OR 9,6:PRINT A$:GOTO 10720   <093>
10750 !CURS OR 0,4:PRINT A$;A$;A$;A$     <211>
10760 £END                               <209>
```

Listing 2. »Türme von Hanoi«; Achtung: Die Befehle CURSOR, CLRSCR, WITKEY und COLOR müssen zusammengeschrieben werden (ohne <SPACE>)! Bitte mit dem Checksummer (Seite 159) ohne Rekursiv-Basic eingeben.

```
50000 REM *******************           <209>
50010 REM * LIBRARY V 1.0     *          <011>
50020 REM * ZUSAMMENGESTELLT *           <023>
50030 REM *        VON:       *          <172>
50040 REM *   THOMAS  KOLBE   *          <085>
50050 REM *******************           <003>
50060 :                                  <251>
50100 £PROC CLS:PRINT"{CLR}";:£END       <000>
50110 :                                  <045>
50150 £PROC CURS OR X,Y                  <010>
50160 POKE 781,Y:POKE 782,X:POKE 783,PEEK(
      783)AND 254:SYS 65520              <110>
50170 £END                               <246>
50180 :                                  <117>
50200 £PROC PRTAT X,Y,A$:!CURS OR X,Y:PRIN
      T A$;:£END                         <124>
50210 :                                  <147>
50220 £PROC WAIT KEY #ZQ$,ER$,MODUS:£VAR I
      :IF MODUS THEN PRINT"{RVSON,SPACE,RV
      OFF,LEFT}";                        <022>
50230 GET ZQ$:IF ZQ$=""THEN 50230        <238>
50235 IF ER$=""THEN 50245                <062>
50240 !POS ITI ON ZQ$,ER$,I:IF I=0 THEN 50
      230                                <198>
50245 IF MODUS THEN PRINT ZQ$;           <086>
50250 £END                               <072>
50260 :                                  <197>
50300 £PROC POS ITI ON A$,B$,#ZQ         <007>
50310 FOR ZQ=1 TO(LEN(B$)-LEN(A$)+1)     <032>
50320 IF A$<>MID$(B$,ZQ,LEN(A$))THEN:NEXT
      ZQ:ZQ=0                            <055>
50330 £END                               <152>
50340 :                                  <021>
50350 £PROC CLR EOL:£VAR I               <140>
50360 I=40-PEEK(211)                     <250>
50370 PRINT LEFT$("{40SPACE}",I);:£END   <136>
50380 :                                  <061>
50400 £PROC LINIE A:FOR A=A TO 1 STEP-1:PR
      INT"@";:NEXT A:£END                <100>
50410 :                                  <091>
50450 £PROC COL OR RA,HI:POKE 53280,RA:POK
      E 53281,HI:£END                    <214>
50460 :                                  <143>
50500 £PROC CENTER A$                    <243>
50510 !PRTAT INT((40-LEN(A$))/2),PEEK(214)
      ,A$:£END                           <067>
50520 :                                  <203>
50550 £PROC FETCH #ZY$,L:£VAR I,T$:ZY$="" <054>
50560 !LINIE L:FOR I=1 TO L:PRINT"{LEFT}";
      :NEXT I                            <116>
50570 PRINT"{RVSON,SPACE,RVOFF,LEFT}";   <178>
50575 !WAIT KEY T$,"",0                  <235>
50580 IF T$>=" "AND T$<="←"AND L>LEN(ZY$)T
      HEN PRINT T$;:ZY$=ZY$+T$:GOTO 50570 <213>
50590 IF T$=CHR$(20)AND LEN(ZY$)>0 THEN ZY
      $=LEFT$(ZY$,LEN(ZY$)-1):PRINT"{LEFT,
      SPACE}@{2LEFT}";                   <074>
50600 IF T$=CHR$(20)THEN 50570           <132>
50610 IF T$<>CHR$(13)THEN 50575          <026>
50620 PRINT"@":£END                      <136>
50630 :                                  <057>
50650 £PROC L PRINT SEK,A$:OPEN 1,4,SEK:PR
      INT#1,A$:CLOSE 1:£END              <174>
50660 :                                  <087>
50670 £PROC STATUS #QN,#QE$,#QT,#QS:CLOSE
      15:OPEN 15,8,15                    <053>
50680 INPUT#15,QN,QE$,QT,QS:CLOSE 15:£END <101>
50690 :                                  <119>
50700 £PROC BEEP H,L:£VAR I:POKE 54296,15:
      POKE 54295,0:POKE 54272,H-INT(H/256)
      *256                               <137>
50710 POKE 54273,H/256:POKE 54277,26:POKE
      54278,230:POKE 54276,33            <099>
50720 FOR I=1 TO L:NEXT I:POKE 54276,32:£E
      ND                                 <075>
50730 :                                  <159>
50740 £PROC CATA LOG:£VAR A$,B$:CLOSE 3  <231>
50750 OPEN 3,8,0,"$":GET#3,A$,A$         <078>
50760 GET#3,A$,A$:IF ST=64 THEN CLOSE 3:£E
      ND                                 <063>
50770 GET#3,A$,B$:PRINT MID$(STR$(ASC(A$+C
      HR$(0))+256*ASC(B$+CHR$(0))),2)" "; <201>
50780 GET#3,A$:PRINT A$;:IF A$<>""THEN 507
      80                                 <245>
50790 PRINT:GOTO 50760                   <165>
```

Listing 3. »LIBRARY« Auch hier müssen die neuen Befehle CURSOR, WAITKEY, POSITION, CLREOL, COLOR LPRINT und CATALOG zusammengeschrieben werden. Bitte mit dem Checksummer (Seite 159) ohne Rekursiv-Basic eingeben.

```
Name : merger                 02c0 02d8

02c0 : 20 fd ae 20 d4 e1 a4 2e   da
02c8 : a5 2d 38 e9 02 aa b0 01   89
02d0 : 88 a9 00 85 0a 4c 75 e1   7a
```

Listing 4. »MERGER« zum Einbinden der Library von Rekursiv-Basic in eigene Programme

Maschinenroutinen in Basic-Zeilen

Die Verbindung von Basic-Programmen mit Routinen in Maschinensprache ist eine der reizvollsten Programmiertechniken auf dem C64. Mit einem neuartigen Hilfsprogramm zeigen wir Ihnen eine interessante Alternative zum Nachladen von Maschinenroutinen.

Üblicherweise werden Maschinenprogramme, die von Basic-Programmen aufgerufen werden, nachträglich geladen. Dieses »Nachladen« von Maschinenroutinen hat jedoch einige Nachteile:
- Der Start der Routinen liegt fast immer bei 828 ($033C) oder 49152 ($C000). Will man mehrere Routinen gleichzeitig verwenden, müssen sie oftmals auf andere Adreßbereiche umgeschrieben werden.
- Bei einem Reset verschwinden Programme aus dem Kassettenpuffer ($033C - 03FB). Sie müssen nachgeladen werden, selbst wenn das Basic-Programm durch einen OLD-Befehl, wie manche Basic-Erweiterungen ihn besitzen, gerettet werden kann.
- Das Nachladen der Routinen ist eine Prozedur, die zusätzliche Zeit in Anspruch nimmt und das Programmieren einengt.
- Wird eine Routine vergessen oder durch POKEs gestört, kann es zu Abstürzen kommen, die unter Umständen stundenlange Arbeit unwiederbringlich vernichten.

Diesen Ärger gibt es nicht mehr, wenn die Unterroutinen, die für ein Programm benötigt werden, am Anfang des Basic-Programms stehen und infolgedessen mit ihm geladen werden. Mit »SYS Variable« und gegebenfalls weiteren Parametern erreichen Sie diese Routinen jederzeit.

Das Programm »MPRG IM BASIC« übernimmt diese Einbindung beliebiger kurzer Maschinenroutinen in Basic-Zeilen. Dabei sind jedoch einige Vorgaben zu beachten:

REM-Zeilen machen's möglich

Der Basic-Interpreter des C64 verarbeitet beim Programmieren, Laden und Listen jeweils Zeilen mit bis zu 255 Zeichen. Darin enthalten sind zwei Byte als Zeiger auf den Beginn der nächsten Zeile, zwei Byte für die Zeilennummer und das Schlußbyte ($00).

Beim Ablauf eines Basic-Programms wird bei REM die nächste Zeile angesprungen. Folglich können hinter einem REM, das am Anfang einer Zeile steht, 249 beliebige Bytes stehen, die nicht vom Basic-Interpreter ausgeführt werden und somit auch nicht der üblichen Basic-Syntax des C64 entsprechen müssen. Anstelle der üblichen Bemerkungen im Klartext können diese Bytes folglich auch zu einem Maschinenprogramm gehören. Dieser Trick wurde durch den Basic-Einzeiler-Wettbewerb im 64'er-Magazin sehr beliebt.

Mit solchen mehr oder weniger langen REM-Zeilen, die Maschinenprogramme anstelle von Kommentaren enthalten, wäre also der Platz für Routinen im Basic-Programm geschaffen. Für diese Routinen müssen aber noch zwei Regeln beachtet werden:

1. Es darf kein Null-Byte vorkommen, da eine Null das Zeilenende markiert und bei Einbindung oder Löschen anderer Zeilen die REM-Zeile hier abgeschnitten wird. Durch INC, DEC oder Laden aus der Zeropage läßt sich das Auftreten von Null-Bytes jedoch mit vertretbarem Aufwand umgehen.

2. Wer die REM-Zeilen wahlweise einsetzen will, der darf - abgesehen vom Aufruf der ROM-Routinen - nur bedingte Verzweigungen (Branches) verwenden.

Eine direkte Änderung dieser REM-Zeilen auf dem Bildschirm ist nicht allein wegen einer möglichen Überlänge, sondern auch wegen der eigenartigen Ausgabe unmöglich (Bild 1). Wenn nämlich der Interpreter beim Listen das momentane Byte keinem Zeichen zuordnen kann, behandelt er es als Token und druckt das entsprechende Basic-Befehlswort aus. Zum Teil führt er sogar Steuerzeichen aus (zum Beispiel setzt der Wert 31 die Schriftfarbe auf Blau).

Programmbeschreibung

Verwirrende optische Effekte, die teilweise wie Systemabstürze wirken, sind die Folge. <RUN/STOP RESTORE> bringt alles wieder in einen geordneten Zustand.

Wer hingegen einen Listschutz vortäuschen will, kann das Resultat dieser Pseudo-Abstürze so stehen lassen und ab der ersten Basic-Programmzeile listen. Eleganter ist es aber, eine »KORREKTURZEILE« (siehe unten) anzuhängen, die den Bildschirm »aufräumt«. Beim Wert 204 ($CC) jedoch stoppt das Listen aufgrund eines kleinen Fehlers im Betriebssystem des C64 unweigerlich und es erscheint die Meldung SYNTAX ERROR.

Das Basic-Programm »MPRG IM BASIC« (Listing 1) verändert sich während des Ablaufs selbständig. Daher darf es nicht geändert werden. Jede unbedachte Änderung führt zur Fehlfunktion! Der Originalzustand im Speicher besteht nur vor dem ersten Start.

Deshalb muß dieses Programm mit dem MSE eingegeben werden, so daß es aufs Byte genau stimmt. Dies gilt auch für die weiteren Hilfsprogramme:

»MERGE« (Listing 2) ist eine Routine zum Anhängen eines Basic-Programms an das im Speicher befindliche Programm. Diese Routine eignet sich hervorragend zu einem ersten Test von »MPRG IM BASIC« und wird im weiteren für alle folgenden Operationen als Hilfsroutine benötigt.

»STARTADRESSEN« (Listing 3) ist ein Basic-Programm zur Ermittlung der Einsprungadressen mehrerer Routinen in REM-Zeilen. Es beginnt mit der Zeile 60000 (Start mit RUN 60000), numeriert die Zeilen fortlaufend ab 1, gibt die Adressen dezimal auf dem Bildschirm und Drucker (Geräteadresse 4) aus und vermag sich nach geleisteter Arbeit selbst zu löschen. Dieses Programm muß ebenfalls mit MSE abgetippt werden, weil sonst aufgrund einer unterschiedlichen Programmlänge Fehler auftreten könnten. Änderungen sind also ausgeschlossen.

»KORREKTURZEILE« (Listing 4), ein Einzeiler, enthält hinter einem REM die Bytes, die nach dem Listen der REM-Zeilen den Normalzustand des Bildschirms wieder herstellen.

Die Anwendung der Programme

Nach dem Start mit RUN gibt »MPRG IM BASIC« eine Kurzinformation aus und möchte wissen, ob sich das in eine Basic-Zeile umzusetzende Maschinenprogramm, dessen Startadresse beliebig ist, auf der Diskette im Laufwerk befindet. Es wird dann nach dem Programmnamen gefragt. Sie können diesen mit den von Floppy-Befehlen zugelassenen Jokern (»*« und »?«) abkürzen. Nach der Eingabe wird das Programmfile geöffnet. Der Status der Dis-

Bild 1. So sehen die REM-Zeilen mit den Maschinenprogrammen nach einem LIST-Befehl aus

kette wird angezeigt. Bei Fehlermeldungen ist die Leertaste zu drücken. Bei der darauf folgenden Eingabe können Sie eine Zeilennummer von 1 bis 99 wählen.

Erst jetzt wird das Maschinenprogramm geladen und überprüft. Es erscheint ein Vorschlag für den Namen der REM-Zeile, der sich aus Zeilennummer und eingegebenem File-Namen zusammensetzt. Diesen Vorschlag können Sie unverändert übernehmen oder eigene Eingaben tätigen.

Vor dem Speichern ist Gelegenheit zum Diskettenwechsel. Es wird auch hier der Diskettenstatus angezeigt, so daß ein irrtümliches Überschreiben eines Programms verhindert wird. Das Speichern ist mehrmals möglich. Nun können Sie die nächste Routine bearbeiten lassen oder erst das Umwandlungsergebnis in eine REM-Zeile ansehen. Entschließen Sie sich zu letzterem, wird das Programm gelöscht und die neue REM-Zeile geladen. Die Startadresse für den SYS-Aufruf liegt bei 2054.

Zum Einbinden einer REM-Zeile mit Maschinenprogramm in Ihr Basic-Programm gehen Sie wie folgt vor:

1. Wandeln Sie zunächst die MERGE-Routine (Listing 2) wie gerade beschrieben in eine REM-Zeile um und speichern Sie diese. Anschließend laden Sie diese REM-Zeile und starten sie mit SYS 2054. Es erscheint die Frage nach dem Namen des noch zu ladenden Programms.
2. Nacheinander die gewünschten Routinen mit Hilfe von MERGE laden, wobei die Reihenfolge hier beliebig ist.
3. Wenn man alle gewünschten REM-Zeilen für sein Basic-Programm geladen hat, hängt man mit MERGE die REM-Zeile »Korrekturzeile« (umgewandeltes Listing 4) an. Nun merken Sie sich bitte die Anzahl aller REM-Zeilen inklusive MERGE und Korrekturzeile.
4. Das Basic-Hauptprogramm mit MERGE nachladen. Die erste Zeilennummer muß größer als die Anzahl der zuvor geladenen Zeilen sein. Die höchste Zeilennummer hat kleiner als 60000 zu sein.
5. Die Hilfsroutine »STARTADRESSEN« mit MERGE anhängen und mit RUN 60000 starten. Nach dem Ausdruck der neu ermittelten Einsprungadressen können Sie dieses Programm löschen. Prüfen Sie jedoch zuvor, ob Sie die MERGE-Routine noch benötigen. Sie steht jetzt in Zeile 1. Sobald Sie eine REM-Zeile gelöscht haben, müssen Sie die Routine »STARTADRESSEN« erneut mit RUN 60000 starten.
6. Den REM-Zeilen sollte eine Zeile folgen, in der Basic-Variablen als Einsprungadressen (wie mit »STARTADRESSEN« ermittelt) für die Routinen definiert werden, zum Beispiel ME=2054 für MERGE. SYS ME ruft dann die MERGE-Routine auf.

Auf der Programmservice-Diskette befindet sich neben den Listings 1 bis 4 noch eine Auswahl von zwölf weiteren nützlichen Routinen in Maschinensprache. Zusätzlich wurden diese gleich in Basic-Zeilen umgesetzt und sind außerdem in einem eigenen Demonstrationsprogramm enthalten, welches auch die Anwendung jeder Routine inklusive der eventuell nötigen Parameter aufzeigt.

Wir haben Ihnen hier eine praktikable und interessante Programmiertechnik zur Verbindung von Basic und Maschinensprache aufgezeigt und sind auf Ihre Reaktionen und vor allem auf die mit dieser Methode entwickelten Programme gespannt.

Sollten Sie also Gefallen an dieser Programmiertechnik finden und eigene kleine Routinen entwickeln, können Sie uns diese jederzeit zusenden.

(Axel Hohlfeld/Florian Müller)

```
Name : mprg im basic      0801 10ba
-----------------------------------
0801 : 44 08 0a 00 8d 35 34 30  80
0809 : 3a 99 22 11 11 20 45 53  88
0811 : 20 57 45 52 44 45 4e 20  60
0819 : 50 52 4f 47 52 41 4d 4d  4e
0821 : 45 20 49 4e 20 4d 41 53  ab
0829 : 43 48 49 4e 45 4e 53 50  61
0831 : 52 41 43 48 45 20 4d 49  1b
0839 : 54 20 42 49 53 20 5a 55  a1
0841 : 22 3b 00 91 08 14 00 99  87
0849 : 22 20 32 34 39 20 42 59  df
0851 : 54 45 53 20 49 4e 20 42  2d
0859 : 41 53 49 43 2d 5a 45 49  4c
0861 : 4c 45 4e 20 20 20 20 56  18
0869 : 45 52 50 41 43 4b 54 20  34
0871 : 55 4e 44 20 41 55 54 4f  b1
0879 : 4d 41 54 49 53 43 48 20  56
0881 : 41 55 46 20 44 49 53 4b  75
0889 : 45 54 54 45 20 22 3b 00  b6
0891 : d6 08 1e 00 99 22 20 20  5e
0899 : 47 45 53 50 45 49 43 48  9e
08a1 : 45 52 54 2e 22 2c 2c 2c  77
08a9 : 22 11 20 49 48 52 20 53  c3
08b1 : 54 41 52 54 20 4c 49 45  d9
08b9 : 47 54 20 4a 45 57 45 49  32
08c1 : 4c 53 20 42 45 49 20 81  29
08c9 : 53 59 53 32 30 35 34 9a  97
08d1 : 2e 20 20 22 00 00 09 28  d0
08d9 : 00 99 22 11 11 11 20 42  ef
08e1 : 49 54 54 45 20 9f 53 48  ef
08e9 : 49 46 54 9a 20 44 52 55  d6
08f1 : 45 43 4b 45 4e 2e 22 3a  a7
08f9 : 92 36 35 33 2c 31 00 43  2d
0901 : 09 64 00 8d 35 34 30 3a  18
0909 : 99 2c 22 11 4d 41 58 49  36
0911 : 4d 41 4c 20 32 34 39 20  00
0919 : 5a 45 49 43 48 45 4e 22  fd
0921 : a6 32 32 29 22 53 54 41  23
0929 : 52 54 20 4d 49 54 20 53  b5
0931 : 59 53 32 30 35 34 11 11  22
0939 : 22 3a 9f 32 2c 38 2c 31  3e
0941 : 35 00 7c 09 6e 00 99 22  48
0949 : 20 49 53 54 20 44 41 53  3d
0951 : 20 50 52 47 20 41 55 46  05
0959 : 20 44 49 53 4b 45 54 54  31
0961 : 45 20 28 4a 2f 4e 29 3f  92
0969 : 22 3a 8d 35 30 30 3a 8b  37
0971 : 41 24 b2 22 4e 22 a7 35  b4
0979 : 32 30 00 b2 09 78 00 85  79
0981 : 22 20 4e 41 4d 45 20 44  77
0989 : 45 53 20 50 52 4f 47 52  eb
0991 : 41 4d 4d 53 22 3b 4e 41  ee
0999 : 24 3a 9f 31 2c 38 2c 38  8e
09a1 : 2c 4e 41 24 aa 22 2c 50  d6
09a9 : 2c 52 22 3a 8d 35 36 30  8a
09b1 : 00 dc 09 82 00 8b 41 b2  79
09b9 : 36 32 a7 97 31 39 38 2c  fb
09c1 : 30 3a 92 31 39 38 2c 31  42
09c9 : 3a 8d 35 39 30 3a 8d 35  b4
09d1 : 39 30 3a a0 31 3a 89 31  32
09d9 : 32 30 00 0a 0a 8c 00 99  9d
09e1 : 3a 99 3a 8b c9 28 4e 41  81
09e9 : 24 2c 31 29 b2 22 2a 22  be
09f1 : a7 41 b2 c3 28 4e 41 24  a0
```

Listing 1. »MPRG IM BASIC« ist zwar ein Basic-Programm, muß jedoch mit dem MSE (Seite 159) eingegeben werden und darf keinesfalls geändert werden

```
09f9 : 29 ab 31 3a 4e 41 24 b2  70
0a01 : c8 28 4e 41 24 2c 41 29  94
0a09 : 00 4b 0a 96 00 85 22 91  dc
0a11 : 20 42 41 53 49 43 2d 5a  25
0a19 : 45 49 4c 45 4e 4e 55 4d  06
0a21 : 4d 45 52 20 20 20 20 9d  68
0a29 : 9d 9d 9d 22 3b 5a 4e 24  49
0a31 : 3a 8b c6 28 5a 4e 24 29  e2
0a39 : b1 35 37 b0 c6 28 5a 4e  1c
0a41 : 24 29 b3 34 38 a7 31 35  5d
0a49 : 30 00 65 0a a0 00 5a 4e  24
0a51 : b2 c5 28 5a 4e 24 29 3a  5a
0a59 : 8b 5a 4e b3 31 30 30 a7  c0
0a61 : 31 38 30 00 a6 0a aa 00  20
0a69 : 99 22 91 20 05 4e 49 43  ea
0a71 : 48 54 20 47 52 4f 45 53  30
0a79 : 53 45 52 20 41 4c 53 20  0b
0a81 : 39 39 21 9a 20 20 20 20  b6
0a89 : 20 20 20 20 20 20 20 20  89
0a91 : 22 3a 97 31 39 38 2c 30  43
0a99 : 3a 92 31 39 38 2c 31 3a  ae
0aa1 : 89 31 35 30 00 c2 0a b4  be
0aa9 : 00 53 56 24 b2 c8 28 22  cf
0ab1 : 5a 4e 24 aa 22 20 22 aa  92
0ab9 : 4e 41 24 29 2c 31 36 29  4e
0ac1 : 00 f9 0a be 00 4e 31 b2  b5
0ac9 : 32 30 35 34 3a 4e 32 b2  2b
0ad1 : 32 33 30 32 3a 97 32 30  79
0ad9 : 34 39 2c 30 3a 97 32 30  45
0ae1 : 35 30 2c 39 3a 97 32 30  ea
0ae9 : 35 31 2c 5a 4e 3a 97 32  87
0af1 : 30 35 33 2c 31 34 33 00  90
0af9 : 20 0b c8 00 a1 23 31 2c  21
0b01 : 41 24 3a a1 23 31 2c 41  06
0b09 : 24 3a 99 22 11 9f 20 4c  1c
0b11 : 4f 41 44 49 4e 47 20 22  1f
0b19 : 4e 41 24 22 9a 22 00 56  bd
0b21 : 0b d2 00 81 49 b2 4e 31  8b
0b29 : a4 4e 32 3a a1 23 31 2c  19
0b31 : 41 24 3a 41 b2 c6 28 41  c0
0b39 : 24 aa c7 28 30 29 29 3a  0f
0b41 : 97 49 2c 41 3a 8b 41 b2  1b
0b49 : 30 a7 49 b2 49 ab 31 3a  21
0b51 : 89 32 34 30 00 62 0b dc  ff
0b59 : 00 8b 53 54 a7 32 34 30  bb
0b61 : 00 9b 0b e6 00 82 3a 99  ff
0b69 : 22 11 20 05 50 52 4f 47  20
0b71 : 52 41 4d 4d 20 5a 55 20  cb
0b79 : 4c 41 4e 47 21 9a 22 3a  c6
0b81 : 97 31 39 38 2c 30 3a 92  59
0b89 : 31 39 38 2c 31 3a 41 24  1d
0b91 : b2 22 4a 22 3a 89 34 31  4e
0b99 : 30 00 db 0b f0 00 42 5a  ee
0ba1 : b2 49 3a a0 31 3a 99 22  2a
0ba9 : 11 20 53 50 45 49 43 48  e5
0bb1 : 45 52 4e 20 41 55 46 20  cf
0bb9 : 44 49 53 4b 45 54 54 45  b3
0bc1 : 20 22 2c 22 22 20 41 4c  44
0bc9 : 53 9f 20 22 53 56 24 22  f5
0bd1 : 9a 20 28 4a 2f 4e 29 3f  57
0bd9 : 22 00 f8 0b 04 01 8d 35  84
0be1 : 30 30 3a 99 3a 8b 41 24  38
0be9 : b2 22 4a 22 a7 99 22 11  75
0bf1 : 22 3a 89 32 39 30 00 36  5a
0bf9 : 0c 0e 01 97 31 39 2c 36  3a
0c01 : 34 3a 85 22 20 4e 45 55  2c
0c09 : 45 52 20 50 52 4f 47 52  eb
0c11 : 41 4d 4d 4e 41 4d 45 3a  1e
0c19 : 20 22 3b 41 24 3a 97 31  16
0c21 : 39 2c 30 3a 53 56 24 b2  a2
0c29 : c8 28 41 24 2c 31 36 29  52
0c31 : 3a 99 22 11 00 6b 0c 22  b2
0c39 : 01 99 22 20 44 49 53 4b  06
0c41 : 45 54 54 45 20 45 49 4e  5c
0c49 : 47 45 4c 45 47 45 4e 20  07
0c51 : 55 4e 44 20 9f 53 48 49  2b
0c59 : 46 54 9a 2e 22 3a 92 36  e0
0c61 : 35 33 2c 31 3a 8d 35 39  b8
0c69 : 30 30 00 8a 0c 2c 01 53  57  84
0c71 : 24 b2 53 56 24 aa c7 28  95
0c79 : 33 34 29 aa 22 2c 38 3a  3f
0c81 : 22 aa c7 28 31 34 33 29  c3
0c89 : 00 b8 0c 36 01 4e 56 b2  f0
0c91 : c3 28 53 57 24 29 3a 81  a0
0c99 : 49 b2 31 a4 4e 56 3a 97  cc
0ca1 : 33 33 37 31 aa 49 2c c6  95
0ca9 : 28 ca 28 53 57 24 2c 49  85
0cb1 : 2c 31 29 29 3a 82 00 f9  91
0cb9 : 0c 40 01 41 b2 42 5a aa  4a
0cc1 : 32 3a 97 32 30 34 39 2c  1e
0cc9 : 41 af 32 35 35 3a 97 32  fd
0cd1 : 30 35 30 2c 41 ad 32 35  e2
0cd9 : 36 3a 97 32 35 31 2c 28  36
0ce1 : 41 aa 32 29 af 32 35 35  f5
0ce9 : 3a 97 32 35 32 2c 28 41  ca
0cf1 : aa 32 29 ad 32 35 36 00  5a
0cf9 : 26 0d 4a 01 97 41 ab 31  ed
0d01 : 2c 30 3a 97 41 2c 30 3a  72
0d09 : 97 41 aa 31 2c 30 3a 97  6e
0d11 : 34 35 2c c2 28 32 35 31  8f
0d19 : 29 3a 97 34 36 2c c2 28  ec
0d21 : 32 35 32 29 00 45 0d 54  a7
0d29 : 01 94 22 2a 2a 2a 2a 2a  33
0d31 : 2a 2a 2a 2a 2a 2a 2a 2a  31
0d39 : 2a 2a 2a 2a 2a 2a 2a 2a  39
0d41 : 2a 2a 2a 00 7e 0d 5e 01  d7
0d49 : 97 34 35 2c 31 38 36 3a  ef
0d51 : 97 34 36 2c 31 36 3a 97  f2
0d59 : 32 30 34 39 2c 30 3a 97  34
0d61 : 32 30 35 30 2c 39 3a 97  a3
0d69 : 32 33 30 34 2c 36 37 3a  8d
0d71 : 97 32 33 30 35 2c 39 3a  02
0d79 : 8d 35 36 30 30 00 ab 0d 68  97
0d81 : 01 91 41 aa 31 89 33 39  8f
0d89 : 30 3a 8b 41 b2 33 33 a7  c2
0d91 : 97 31 39 38 2c 30 3a 92  69
0d99 : 31 39 38 2c 31 3a 97 31  a0
0da1 : 39 38 2c 30 3a 89 33 38  35
0da9 : 35 00 dd 0d 72 01 99 22  d1
0db1 : 20 81 55 45 42 45 52 53  ce
0db9 : 43 48 52 45 49 42 45 4e  b6
0dc1 : 3f 9a 20 28 4a 2f 4e 29  04
0dc9 : 9a 22 3a 8d 35 30 30 3a  bf
0dd1 : 8b 41 24 b2 22 4e 22 a7  c9
0dd9 : 33 38 35 00 f9 0d 7c 01  72
0de1 : 53 57 24 b2 22 40 30 3a  99
0de9 : 22 aa 53 57 24 3a 8d 35  d5
0df1 : 39 30 3a 89 33 31 30 00  80
0df9 : 0e 0e 81 01 8d 35 39 30  57
0e01 : 3a 99 22 91 91 91 22 3a  65
0e09 : 89 32 37 30 00 45 0e 86  ef
0e11 : 01 8d 35 39 30 3a 99 22  cd
0e19 : 20 4e 4f 43 48 4d 41 4c  29
0e21 : 20 53 50 45 49 43 48 45  02
0e29 : 52 4e 20 28 4a 2f 4e 29  59
0e31 : 3f 22 3a 8d 35 30 30 3a  cc
0e39 : 8b 41 24 b2 22 4a 22 a7  11
0e41 : 32 39 30 00 69 0e 90 01  67
0e49 : 99 22 11 9f 20 57 45 49  90
0e51 : 54 45 52 4d 41 43 48 45  60
0e59 : 4e 20 28 4a 2f 4e 29 3f  93
0e61 : 9a 22 3a 8d 35 30 30 00  e2
0e69 : 98 0e 9a 01 8d 35 34 30  83
0e71 : 3a 81 49 b2 4e 31 a4 4e  b2
0e79 : 32 3a 97 49 2c 34 32 3a  79
0e81 : 82 3a 81 49 b2 31 [illegible] 32  55
0e89 : 34 3a 97 33 33 37 31 aa  2e
0e91 : 49 2c 34 32 3a 82 00 af  5b
0e99 : 0e a4 01 97 32 30 35 31  08
0ea1 : 2c 31 30 3a 97 32 33 37  ff
0ea9 : 35 2c 31 34 33 00 ca 0e  42
0eb1 : ae 01 a0 31 3a a0 32 3a  14
0eb9 : 97 31 39 38 2c 30 3a 8b  82
0ec1 : 41 24 b2 22 4a 22 a7 8a  6f
0ec9 : 00 ff 0e b8 01 99 22 11  eb
0ed1 : 11 20 44 49 45 20 4e 45  46
0ed9 : 55 45 20 5a 45 49 4c 45  7f
0ee1 : 20 4c 41 44 45 4e 3f 20  04
0ee9 : 28 4a 2f 4e 29 22 3a 8d  74
0ef1 : 35 30 30 3a 8b 41 24 b2  4a
0ef9 : 22 4e 22 a7 80 00 35 0f  bb
0f01 : c2 01 97 36 33 31 2c 31  c0
0f09 : 34 37 3a 97 36 33 32 2c  78
0f11 : 37 36 3a 97 36 33 33 2c  07
0f19 : 32 30 37 3a 97 36 33 34  d9
0f21 : 2c 38 33 3a 97 36 33 35  e0
0f29 : 2c 38 36 3a 97 36 33 36  ab
0f31 : 2c 33 36 00 5a 0f cc 01  d8
0f39 : 97 36 33 37 2c 34 34 3a  49
0f41 : 97 36 33 38 2c 35 36 3a  81
0f49 : 97 36 33 39 2c 31 33 3a  7d
0f51 : 97 31 39 38 2c 39 3a 80  4d
0f59 : 00 78 0f f4 01 a1 41 24  62
0f61 : 3a 8b 41 24 b2 22 4a 22  df
0f69 : b0 41 24 b2 22 4e.22 a7  86
0f71 : 8d 35 39 30 3a 8e 00 81  08
0f79 : 0f fe 01 89 35 30 30 00  0e
0f81 : c9 0f 08 02 99 22 93 11  2f
0f89 : 05 20 44 41 53 20 4d 41  c6
0f91 : 53 43 48 49 4e 45 4e 50  aa
0f99 : 52 4f 47 52 41 4d 4d 20  a3
0fa1 : 42 49 54 54 45 22 2c 2c  96
0fa9 : 22 11 20 45 52 53 54 20  56
0fb1 : 41 55 46 20 44 49 53 4b  a5
0fb9 : 45 54 54 45 20 20 52 49  d6
0fc1 : 4e 47 45 4e 21 9a 22 00  3d
0fc9 : d2 0f 12 02 a0 32 3a 80  6d
0fd1 : 00 09 10 1c 02 99 c7 28  3a
0fd9 : 31 34 32 29 3b 22 93 11  0b
0fe1 : 9f 20 20 20 42 41 53 49  ab
0fe9 : 43 2d 5a 45 49 4c 45 20  4e
0ff1 : 4d 49 54 20 4d 41 53 43  af
0ff9 : 48 49 4e 45 4e 50 52 4f  71
1001 : 47 52 41 4d 4d 9a 22 00  9e
1009 : 30 10 26 02 99 a3 37 29  f1
1011 : 22 11 28 43 29 20 31 39  f9
1019 : 38 36 20 42 59 20 41 58  09
1021 : 45 4c 20 48 4f 48 4c 46  92
1029 : 45 4c 44 22 3a 8e 00 4f  a0
1031 : 10 30 02 84 32 2c 41 2c  4c
1039 : 41 24 3a 99 22 81 20 53  a3
1041 : 54 41 54 55 53 3a 20 22  c1
1049 : 41 24 22 9a 22 00 69 10  60
1051 : 3a 02 8b 41 b2 30 b0 41  89
1059 : b2 33 33 b0 41 b2 36 32  6f
1061 : b0 41 b2 36 33 a7 8e 00  d0
1069 : 87 10 44 02 97 31 39 38  a2
1071 : 2c 30 3a 92 31 39 38 2c  ac
1079 : 31 3a 41 24 b2 22 4a 22  46
1081 : 3a 89 34 31 30 00 b8 10  b9
1089 : 4e 02 99 22 91 20 20 20  5e
1091 : 20 20 20 20 20 20 20 20  91
1099 : 20 20 20 20 20 20 20 20  99
10a1 : 20 20 20 20 20 20 20 20  a1
10a9 : 20 20 20 20 20 20 20 20  a9
10b1 : 20 20 91 22 3a 8e 00 00  a2
10b9 : 00 52 49 46 54 20 4c 41  f7
```

Listing 1. (Schluß)

```
Name : merge                   033e 038c
----------------------------------------
033e : a0 ff c8 84 0a b9 07 a2  70
0346 : 48 29 7f 20 d2 ff c8 68  28
034e : 10 f3 20 45 ab 20 3f ab  19
0356 : a2 01 20 cf ff c9 0d f0  df
035e : 0d 9d ff 01 e8 e0 11 90  55
0366 : f1 a2 17 4c 37 a4 ca 8a  d1
036e : a6 0a a0 02 20 bd ff a2  b7
0376 : 08 a4 0a 20 ba ff a5 2d  f4
037e : a4 2e 38 e9 02 b0 01 88  3f
0386 : aa a5 0a 4c 75 e1 00 00  75
```

Listing 2. »MERGE« eignet sich sowohl zum Test als auch zur Anwendung als Hilfsroutine. Bitte mit dem MSE eingeben.

```
Name : startadressen           0801 0c9d
----------------------------------------
0801 : 3e 08 60 ea 99 22 93 05  bc
0809 : 11 22 2c 22 53 54 41 52  fc
0811 : 54 41 44 52 45 53 53 45  28
0819 : 4e 20 56 4f 4e 22 2c 2c  f6
0821 : 2c 22 11 9d 9d 4d 2d 50  f0
0829 : 52 4f 47 52 41 4d 4d 45  7d
0831 : 4e 20 49 4d 20 42 41 53  4b
0839 : 49 43 9a 22 00 7f 08 6a  00
0841 : ea 99 22 11 11 20 48 41  58
0849 : 42 45 4e 20 53 49 45 20  9a
0851 : 4d 45 48 52 45 52 45 20  d9
0859 : 4d 2d 50 52 47 20 49 4e  d2
0861 : 20 42 41 53 49 43 2d 5a  75
0869 : 45 49 4c 45 4e 20 4d 49  bc
0871 : 54 20 12 20 4d 45 52 47  35
0879 : 45 20 92 22 3b 00 be 08  76
0881 : 74 ea 99 22 20 56 45 52  83
0889 : 4b 4e 55 45 50 46 54 20  c2
0891 : 55 4e 44 20 44 49 45 53  6d
0899 : 45 53 20 50 52 4f 2d 20  2f
08a1 : 20 47 52 41 4d 4d 20 41  64
08a9 : 4e 47 45 48 41 45 4e 47  fb
08b1 : 54 2c 20 57 45 52 44 45  91
08b9 : 4e 20 22 3b 00 f3 08 7e  c4
08c1 : ea 99 22 44 49 45 20 53  6f
08c9 : 54 41 52 54 2d 20 20 20  72
08d1 : 20 20 41 44 52 45 53 53  1d
08d9 : 45 4e 20 41 55 54 4f 4d  45
08e1 : 41 54 49 53 43 48 20 45  8b
08e9 : 52 4d 49 54 54 45 4c 54  08
08f1 : 2e 00 3a 09 88 ea 99 22  5a
08f9 : 20 5a 55 47 4c 45 49 43  1f
0901 : 48 20 45 52 46 4f 4c 47  94
0909 : 54 20 45 49 4e 45 20 4e  14
0911 : 45 55 45 20 4e 55 4d 4d  b6
0919 : 45 52 49 45 52 55 4e 47  1a
0921 : 20 44 49 45 53 45 52 20  47
0929 : 5a 45 49 4c 45 4e 20 41  cc
0931 : 42 20 31 20 49 4e 22 3b  da
```

```
0939 : 00 54 09 92 ea 99 22 20  3c
0941 : 45 49 4e 45 52 53 43 48  c4
0949 : 52 49 54 54 45 4e 2e 11  81
0951 : 11 22 00 7d 09 9c ea 85  4f
0959 : 22 20 41 4e 5a 41 48 4c  0f
0961 : 20 44 45 52 20 5a 45 49  bb
0969 : 4c 45 4e 20 22 3b 41 5a  a5
0971 : 24 3a 41 5a b2 c5 28 41  ca
0979 : 5a 24 29 00 b8 09 a6 ea  74
0981 : 8b 41 5a b2 30 a7 99 22  85
0989 : 91 20 20 20 20 20 20 20  fa
0991 : 20 20 20 20 20 20 20 20  91
0999 : 20 20 20 20 20 20 20 20  99
09a1 : 20 20 20 20 20 20 20 20  a1
09a9 : 20 20 20 20 20 91 22 3a  71
09b1 : 89 36 30 30 36 30 00 d7  fc
09b9 : 09 b0 ea 99 22 93 11 11  2d
09c1 : 20 9f 57 41 52 54 45 4e  28
09c9 : 20 53 49 45 20 42 49 54  70
09d1 : 54 45 9a 11 22 00 01 0a  cb
09d9 : ba ea 5a b2 30 3a 4e b2  69
09e1 : 31 3a 41 4e b2 32 30 35  31
09e9 : 31 3a 45 4e b2 41 4e aa  16
09f1 : 41 5a ac 32 35 36 3a 86  cc
09f9 : 53 41 28 31 30 30 29 00  46
0a01 : 32 0a c4 ea 99 22 13 22  02
0a09 : 4e 22 9d 2e 20 5a 45 49  12
0a11 : 4c 45 22 3a 97 41 4e 2c  e5
0a19 : 4e 3a 41 4e b2 41 4e aa  62
0a21 : 33 3a 53 41 28 4e 29 b2  6d
0a29 : 41 4e 3a 4e b2 4e aa 31  94
0a31 : 00 49 0a c9 ea 8b 4e b1  39
0a39 : 41 5a a7 99 22 11 11 22  f8
0a41 : 3a 89 36 30 31 33 30 00  41
0a49 : 61 0a ce ea 81 49 b2 41  70
0a51 : 4e a4 45 4e 3a 8b c2 28  68
0a59 : 49 29 b3 b1 30 a7 82 00  a4
0a61 : 7a 0a d8 ea 41 4e b2 49  58
0a69 : aa 33 3a 49 b2 45 4e 3a  68
0a71 : 82 3a 89 36 30 31 30 30  e7
0a79 : 00 a2 0a e2 ea 99 2c 22  1a
0a81 : 9d 22 3b 3a 81 49 b2 31  d5
0a89 : a4 36 3a 8b 53 41 28 49  bb
0a91 : aa 5a 29 a7 99 53 41 28  31
0a99 : 49 aa 5a 29 22 9d 22 3b  01
0aa1 : 00 bc 0a e7 ea 82 3a 5a  df
0aa9 : b2 5a aa 36 3a 99 3a 8b  6a
0ab1 : 5a b3 41 5a a7 36 30 31  d0
0ab9 : 33 30 00 e5 0a ec ea 99  a8
0ac1 : 22 11 20 41 44 52 45 53  2f
0ac9 : 53 45 4e 20 41 55 43 48  b3
0ad1 : 20 41 55 53 44 52 55 43  04
0ad9 : 4b 45 4e 3f 20 28 4a 2f  0d
0ae1 : 4e 29 22 00 fb 0a f6 ea  0e
0ae9 : a1 41 24 3a 8b 41 24 b2  34
0af1 : 22 4e 22 a7 36 30 32 30  c6
0af9 : 30 00 0e 0b 00 eb 8b 41  1e
0b01 : 24 b3 b1 22 4a 22 a7 36  70
0b09 : 30 31 35 30 00 4b 0b 0a  c0
0b11 : eb 99 22 20 57 45 4e 4e  cb
0b19 : 20 44 52 55 43 4b 45 52  e3
0b21 : 20 42 45 52 45 49 54 2c  46
0b29 : 20 44 41 4e 4e 20 05 53  26
0b31 : 48 49 46 54 9a 22 3a 92  03
0b39 : 36 35 33 2c 31 3a 5a b2  10
0b41 : 30 3a 9f 34 2c 34 3a 9d  85
0b49 : 34 00 6f 0b 14 eb 99 2c  1a
0b51 : 3a 81 49 b2 31 a4 36 3a  7a
0b59 : 8b 53 41 28 49 aa 5a 29  89
0b61 : a7 99 53 41 28 49 aa 5a  fe
0b69 : 29 22 9d 22 3b 00 89 0b  3f
0b71 : 19 eb 82 3a 5a b2 5a aa  62
0b79 : 36 3a 99 3a 8b 5a b3 41  57
0b81 : 5a a7 36 30 31 38 30 00  d8
0b89 : 93 0b 1e eb 98 34 3a a0  fc
0b91 : 34 00 dc 0b 28 eb 99 22  ea
0b99 : 11 20 45 49 4e 20 54 49  ff
0ba1 : 50 3a 20 41 44 52 45 53  d1
0ba9 : 53 45 4e 20 49 4e 20 52  63
0bb1 : 45 4d 2d 5a 45 49 4c 45  8e
0bb9 : 4e 20 46 41 53 53 45 4e  53
0bc1 : 20 20 42 5a 57 20 56 41  20
0bc9 : 52 49 41 42 4c 45 4e 20  c1
0bd1 : 5a 55 4f 52 44 4e 45 4e  5c
0bd9 : 21 22 00 1b 0c 32 eb 99  a4
0be1 : 22 11 20 53 4f 4c 4c 20  c7
0be9 : 44 49 45 53 45 53 20 53  a4
0bf1 : 55 43 48 50 52 4f 47 52  65
0bf9 : 41 4d 4d 20 22 2c 2c 22  b1
0c01 : 20 47 45 4c 4f 45 53 43  93
0c09 : 48 54 20 57 45 52 44 45  f1
0c11 : 4e 20 28 4a 2f 4e 29 3f  4b
0c19 : 22 00 2d 0c 3c eb a1 41  34
0c21 : 24 3a 8b 41 24 b2 22 4e  6a
0c29 : 22 a7 80 00 40 0c 46 eb  94
0c31 : 8b 41 24 b3 b1 22 4a 22  76
0c39 : a7 36 30 32 32 30 00 71  d5
0c41 : 0c 50 eb 41 b2 c2 28 34  e3
0c49 : 35 29 aa c2 28 34 36 29  65
0c51 : ac 32 35 36 ab 31 31 37  a2
0c59 : 38 3a 97 32 35 31 2c 41  eb
0c61 : af 32 35 35 3a 97 32 35  b1
0c69 : 32 2c 41 ad 32 35 36 00  5d
0c71 : 9b 0c 5a eb 97 41 ab 32  bd
0c79 : 2c 30 3a 97 41 ab 31 2c  cd
0c81 : 30 3a 97 34 35 2c c2 28  4b
0c89 : 32 35 31 29 3a 97 34 36  65
0c91 : 2c c2 28 32 35 32 29 3a  6d
0c99 : 9c 00 00 00 4e 56 3a 97  e5
```

Listing 3. »STARTADRESSEN« ist ebenfalls mit dem MSE einzugeben und darf nicht geändert werden

```
Name : korrekturzeile      0801 0810
------------------------------------
0801 : 0e 08 63 00 8f 22 0d 9a  60
0809 : 8e 93 92 09 00 00 00 52  cb
```

Listing 4. »KORREKTURZEILE«: Auch diese einzelne Basic-Zeile wird mit dem MSE eingegeben

Der Befehlssatz des 6510

Hier finden Sie, alphabetisch geordnet, eine ausführliche Auflistung aller bekannten Befehle des C64-Prozessors. Dazu gehören auch die »illegalen Opcodes«.

Zuerst ein Wort zu den illegalen Opcodes, die in Tabelle 1 enthalten sind:

Seit Erscheinen des C64 sind einige verschiedene Versionen des Prozessors 6510 gebaut worden. Diese sind untereinander voll kompatibel, was den normalen Befehlssatz aus Tabelle 2 anbetrifft. Die illegalen Opcodes jedoch laufen nicht auf allen Versionen der CPU 6510. Welche Befehle auf welchem Computer eine korrekte Ausführung bewirken, läßt sich nur durch Ausprobieren feststellen. Äußerst hilfreich dabei ist der SMON: Er zeigt einen illegalen Opcode nicht wie die meisten Maschinensprachmonitore durch drei Fragezeichen an, sondern disassembliert den Befehl mit den in Tabelle 1 genannten Abkürzungen. Ein vorangestelltes Sternchen (*) kennzeichnet bei SMON (Sonderheft 8/85) den Befehl als illegalen Opcode (zum Beispiel *AXS).

In Tabelle 3 finden Sie eine Übersicht über die in den beiden anderen Tabellen verwendeten Abkürzungen. (sk)

```
■A11 : AND register with $11
Das X- beziehungsweise Y-Register wird mit $11 AND-
verknüpft und das Ergebnis X- beziehungsweise Y-
indiziert abgelegt

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:

A11 OP,X        9C          ABX         3
A11 OP,Y        9E          ABY         3
--------------------------------------------------------
■AAX : AND akku with X-Register and store akku
Entspricht befehlsfolge:
AND zwischen Akku und X-Register
STA

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:

AAX #OP         8B          IM          2
AAX OP          87          ZP          2
AAX OP,Y        97          ZPY         2
AAX OP          8F          ABS         3
AAX (OP,X)      83          (OP,X)      2
--------------------------------------------------------
■ASR : AND with akku and shift right
Entspricht Befehlsfolge:
AND
LSR

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:

ASR #OP         6B          IM          2
--------------------------------------------------------
■ARR : AND with akku and rotate right
Entspricht Befehlsfolge:
AND
ROR

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:

ARR #OP         4b          IM          2
--------------------------------------------------------
■AXS : AND akku and X-register and subtract from data
Der Wert wird von dem Ergebnis der AND-Verknüpfung
zwischen Akku und X-Register subtrahiert und in das X-
Register geschrieben.

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:

AXS #OP         CB          IM          2
--------------------------------------------------------
■DCP : decrement and compare with akku
Entspricht Befehlsfolge:
DEC
CMP

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:

DCP OP          C7          ZP          2
DCP OP,X        D7          ZPX         2
DCP OP          CF          ABS         3
DCP OP,X        DF          ABX         3
DCP OP,Y        DB          ABY         3
DCP (OP,X)      C3          (ZP,X)      2
DCP (OP),Y      D3          (ZP),Y      2
--------------------------------------------------------
■DOP : double NOP
Folgende Codes wirken wie der NOP-Befehl, sind aber
zwei Byte lang. Das zweite Byte wird dabei
übersprungen.

04, 14, 34, 44, 54, 64, 74, D4, F4, 80, 89, 93
--------------------------------------------------------
■ISC : increment and subtract with carry
Entspricht Befehlsfolge:
INC
SBC

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:

ISC OP          E7          ZP          2
ISC OP,X        F7          ZPX         2
ISC OP          EF          ABS         3
ISC OP,X        FF          ABX         3
ISC OP,Y        FB          ABY         3
ISC (OP,X)      E3          (OP,X)      2
ISC (OP),Y      F3          (OP),Y      2
--------------------------------------------------------
■KIL : killer codes
Folgende Codes bewirken einen Absturz des Prozessors,
dem auch mit einem RUN/STOP-RESTORE nicht mehr
beizukommen ist.

02, 12, 22, 32, 42, 52, 62, 72, 92, B2, D2, F2
--------------------------------------------------------
■LAR : load akku, AND with stackregister, transfer
result to akku, X-register and stackregister
Entspricht Befehlsfolge:
LDA
AND
TAX
TXS

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:

LAR OB,Y        BB          ABY         3
--------------------------------------------------------
■LAX : load to akku and X-register
Entspricht Befehlsfolge:
LDA
TAX

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:

LAX OP          A7          ZP          2
LAX OP,Y        B7          ZPY         2
LAX OP          AF          ABS         3
LAX OP,Y        BF          ABY         3
LAX (OP,X)      A3          (OP,X)      2
LAX (OP),Y      B3          (OP),Y      2
--------------------------------------------------------
■NOP : no operation
Folgende Codes haben wie der Code $EA die NOP-Funktion:

1A, 3A, 5A, 7A, DA, FA
```

Tabelle 1. Die »illegalen Opcodes« des 6510-Prozessors

■ **RLA** : rotate left, AND with akku and store akku
Entspricht Befehlsfolge:
ROL
AND
STA

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
RLA OP	27	ZP	2
RLA OP,X	37	ZPX	2
RLA OP	2F	ABS	3
RLA OP,X	3F	ABX	3
RLA OP,Y	3B	ABY	3
RLA (OP,X)	23	(OP,X)	2
RLA (OP),Y	33	(OP),Y	2

■ **RRA** : rotate right and add with carry
Entspricht Befehlsfolge:
ROR
ADC

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
RRA OP	67	ZP	2
RRA OP,X	77	ZPX	2
RRA OP	6F	ABS	3
RRA OP,X	7F	ABX	3
RRA OP,Y	7B	ABY	3
RRA (OP,X)	63	(OP,X)	2
RRA (OP),Y	73	(OP),Y	2

■ **SLO** : shift left and OR with akku

Entspricht Befehlsfolge:
ASL
ORA

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
SLO OP	Ø7	ZP	2
SLO OP,X	17	ZPX	2
SLO OP	ØF	ABS	3
SLO OP,X	1F	ABX	3
SLO OP,Y	1B	ABY	3
SLO (OP,X)	13	(OP,X)	2
SLO (OP),Y	Ø3	(OP),Y	2

■ **SRE** : shift right and EOR with akku
Entspricht Befehlsfolge:
LSR
EOR

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
SRE OP	47	ZP	2
SRE OP,X	57	ZPX	2
SRE OP	4F	ABS	3
SRE OP,X	5F	ABX	3
SRE OP,Y	5B	ABY	3
SRE (OP,X)	43	(OP,X)	2
SRE (OP),Y	53	(OP),Y	2

■ **TOP** : triple NOP
Folgende Codes wirken wie der NOP-Befehl, sind aber drei Byte lang. Das zweite und das dritte Byte wird dabei übersprungen.

ØC, 1C, 3C, 5C, 7C, DC, FC

Tabelle 1. Die »illegalen Opcodes« des 6510-Prozessors (Schluß)

■ **ADC** : add with carry
addiere Adresseninhalt plus Carry-Flag zum Akkumulator

Flags: N Z C I D V
+ + +   +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
ADC #OP	69	IM	2	2
ADC OP	65	ZP	2	3
ADC OP,X	75	ZPX	2	4
ADC OP	6D	ABS	3	4
ADC OP,X	7D	ABX	3	4
ADC OP,Y	79	ABY	3	4
ADC (OP,X)	61	(ZP,X)	2	6
ADC (OP),Y	71	(ZP),Y	2	5

■ **AND** : AND akku
verknüpfe Speicher mit Akku durch logische UND

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
AND #OP	29	IM	2	2
AND OP	25	ZP	2	3
AND OP,X	35	ZPX	2	4
AND OP	2D	ABS	3	4
AND OP,X	3D	ABX	3	4
AND OP,Y	39	ABY	3	4
AND (OP,X)	21	(ZP,X)	2	6
AND (OP),Y	31	(ZP),Y	2	5

■ **ASL** : arithmetic shift left
schiebe Bits eines Speichers um eine Stelle nach links

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
ASL	ØA	Akku	1	2
ASL OP	Ø6	ZP	2	5
ASL OP,X	16	ZPX	2	6
ASL OP	ØE	ABS	3	6
ASL OP,X	1E	ABX	3	7

■ **BCC** : branch if carry clear
verzweige, falls das Übertragsbit gelöscht ist

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BCC OP	90	REL	2	2

■ **BCS** : branch if carry set
verzweige, falls das Übertragsbit gesetzt ist

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BCS OP	BØ	REL	2	2

■ **BEQ** : branch if equal (to zero)
verzweige, falls das Ergebnis der letzten Operation gleich (Null) war

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BEQ OP	FØ	REL	2	2

■ **BIT** : test bits
verknüpfe Speicher und Akku durch AND, setze entsprechende Flags (Akku wird nicht verändert !)

Flags: N Z C I D V
+ +   +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BIT OP	24	ZP	2	3
BIT OP	2C	ABS	3	4

■ **BMI** : branch if minus
verzweige, falls das Ergebnis der letzten Operation kleiner Null war

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BMI OP	3Ø	REL	2	2

Tabelle 2. Die Befehle des 6510-Prozessors

■ BNE : branch if not equal (to zero)
verzweige, falls das Ergebnis der letzten Operation ungleich (Null) war

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BNE OP	D0	REL	2	2

■ BPL : branch if plus
verzweige, falls das Ergebnis der letzten Operation größer Null war

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BPL OP	10	REL	2	2

■ BRK : break
Programmstop und Sprung über Breakpointer

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BRK	00	–	1	7

■ BVC : branch if overflow clear
verzweige, falls das Überlaufsbit gelöscht ist

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BVC OP	50	REL	2	2

■ BVS : branch if overflow set
verzweige, falls das Überlaufsbit gesetzt ist

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BVS OP	70	REL	2	2

■ CLC : clear carry
lösche das Übertragsbit

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CLC	18	–	1	2

■ CLD : clear decimal mode
lösche das Bit für den Dezimalmodus

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CLD	D8	–	1	2

■ CLI : clear interrupt flag
lösche das Interruptbit (Interrupts nun erlaubt)

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CLI	58	–	1	2

■ CLV : clear overflow flag
lösche das Überlaufbit

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CLV	B8	–	1	2

■ CMP : compare with akku
vergleiche Speicher mit Akkuinhalt

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CMP #OP	C9	IM	2	2
CMP OP	C5	ZP	2	3
CMP OP,X	D5	ZPX	2	4
CMP OP	CD	ABS	3	4
CMP OP,X	DD	ABX	3	4
CMP OP,Y	D9	ABY	3	4
CMP (OP,X)	C1	(ZP,X)	2	6
CMP (OP),Y	D1	(ZP),Y	2	5

■ CPX : compare with X-register
vergleiche Speicherinhalt mit X-Register

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CPX #OP	E0	IM	2	2
CPX OP	E4	ZP	2	3
CPX OP	EC	ABS	3	4

■ CPY : compare with Y-register
vergleiche Speicherinhalt mit Y-Register

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CPY #OP	C0	IM	2	2
CPY OP	C4	ZP	2	3
CPY OP	CC	ABS	3	4

■ DEC : decrement
subtrahiere Eins von Speicherinhalt

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
DEC OP	C6	ZP	2	5
DEC OP,X	D6	ZPX	2	6
DEC OP	CE	ABS	3	6
DEC OP,X	DE	ABX	3	7

■ DEX : decrement X-register
subtrahiere Eins vom Inhalt des X-Registers

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
DEX	CA	–	1	2

■ DEY : decrement Y-register
subtrahiere Eins vom Inhalt des Y-Registers

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
DEY	88	–	1	2

■ EOR : exclusive-or
verknüpfe Akku und Speicher durch logisches EXKLUSIV-ODER

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
EOR #OP	49	IM	2	2
EOR OP	45	ZP	2	3
EOR OP,X	55	ZPX	2	4
EOR OP	4D	ABS	3	4
EOR OP,X	5D	ABX	3	4
EOR OP,Y	59	ABY	3	4
EOR (OP,X)	41	(ZP,X)	2	6
EOR (OP),Y	51	(ZP),Y	2	5

■ **INC** : increment
addiere Eins zu Speicherinhalt

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
INC OP	E6	ZP	2	5
INC OP,X	F6	ZPX	2	6
INC OP	EE	ABS	3	6
INC OP,X	FE	ABX	3	7

■ **INX** : increment X-register
addiere Eins zu X-Registerinhalt

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
INX	E8	–	1	2

■ **INY** : increment Y-register
addiere Eins zu Y-Registerinhalt

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
INY	C8	–	1	2

■ **JMP** : jump
springe zu Adresse

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
JMP OP	4C	ABS	3	3
JMP (OP)	6C	IND	3	5

■ **JSR** : jump subroutine
Springe in Unterprogramm

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
JSR OP	20	ABS	3	6

■ **LDA** : load akku
schreibe Wert in Akku

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
LDA #OP	A9	IM	2	2
LDA OP	A5	ZP	2	3
LDA OP,X	B5	ZPX	2	4
LDA OP	AD	ABS	3	4
LDA OP,X	BD	ABX	3	4
LDA OP,Y	B9	ABY	3	4
LDA (OP,X)	A1	(ZP,X)	2	6
LDA (OP),Y	B1	(ZP),Y	2	5

■ **LDX** : load X-register
schreibe Wert ins X-Register

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
LDX #OP	A2	IM	2	2
LDX OP	A6	ZP	2	3
LDX OP,Y	B6	ZPY	2	4
LDX OP	AE	ABS	3	4
LDX OP,Y	BE	ABY	3	4

■ **LDY** : load Y-register
schreibe Wert ins Y-Register

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
LDY #OP	A0	IM	2	2
LDY OP	A4	ZP	2	3
LDY OP,X	B4	ZPX	2	4
LDY OP	AC	ABS	3	4
LDY OP,X	BC	ABX	3	4

■ **LSR** : logical shift right
bitweises Rechtsschieben eines Speicherinhalts (Bit 0 wird ins Carry-Flag geschoben, Bit 7 wird auf Null gesetzt)

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
LSR	4A	Akku	1	2
LSR OP	46	ZP	2	5
LSR OP,X	56	ZPX	2	6
LSR OP	4E	ABS	3	6
LSR OP,X	5E	ABX	3	7

■ **NOP** : no operation
keine Ausführung (Dummy-Befehl)

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
NOP	EA	–	1	2

■ **ORA** : OR akku
verknüpfe Speicherinhalt und Akku durch logisches ODER

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
ORA #OP	09	IM	2	2
ORA OP	05	ZP	2	3
ORA OP,X	15	ZPX	2	4
ORA OP	0D	ABS	3	4
ORA OP,X	1D	ABX	3	4
ORA OP,Y	19	ABY	3	4
ORA (OP,X)	01	(ZP,X)	2	6
ORA (OP),Y	11	(ZP),Y	2	5

■ **PHA** : push akku
schiebe Akkuinhalt auf Stack

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
PHA	48	–	1	3

■ **PHP** : push processor-status
schiebe Statusregister auf Stack

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
PHP	08	–	1	3

Tabelle 2. Die Befehle des 6510-Prozessors (Fortsetzung)

■ PLA : pull akku
lade Akku mit oberstem Stackbyte

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
PLA	68	–	1	3

■ PLP : pull processor-status
lade Statusregister mit oberstem Stackbyte

Flags: N Z C I D V
+ + + + + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
PLP	28	–	1	4

■ ROL : rotate left
rotiere Speicherinhalt um ein Bit nach links (Bit 7 kommt ins Carryflag, Inhalt des Carry-Flags kommt ins Bit Ø)

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
ROL	2A	Akku	1	2
ROL OP	26	ZP	2	5
ROL OP,X	36	ZPX	2	6
ROL OP	2E	ABS	3	6
ROL OP,X	3E	ABX	3	7

■ ROR : rotate right
rotiere Speicherinhalt um ein Bit nach rechts (Bit Ø kommt ins Carryflag, Inhalt des Carryflags kommt ins Bit 7)

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
ROR	6A	Akku	1	2
ROR OP	66	ZP	2	5
ROR OP,X	76	ZPX	2	6
ROR OP	6E	ABS	3	6
ROR OP,X	7E	ABX	3	7

■ RTI : return from interrupt
nach Ausführen eines Interupt normales Programm weiter abarbeiten

Flags: N Z C I D V
wie vor Ausführung des Interrupts

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
RTI	4Ø	–	1	6

■ RTS : return from subroutine
Rücksprung aus Unterprogramm

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
RTS	6Ø	–	1	6

■ SBC : subtract with carry
subtrahiere Speicherinhalt vom Akku unter Berücksichtigung des Vorzeichens

Flags: N Z C I D V
+ + + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
SBC #OP	E9	IM	2	2
SBC OP	E5	ZP	2	3
SBC OP,X	F5	ZPX	2	4
SBC OP	ED	ABS	3	4
SBC OP,X	FD	ABX	3	4
SBC OP,Y	F9	ABY	3	4
SBC (OP,X)	E1	(ZP,X)	2	6
SBC (OP),Y	F1	(ZP),Y	2	5

■ SEC : set carry
setze das Übertragsflag auf Eins

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
SEC	38	–	1	2

■ SED : set decimal mode
setze das Dezimal-Modus-Flag auf Eins

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
SED	F8	–	1	2

■ SEI : set interrupt
setze das Interruptflag auf Eins (es werden keine Interrupts mehr erlaubt)

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
SEI	78	–	1	2

■ STA : store akku
schreibe Akkuinhalt in Speicher

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
STA OP	85	ZP	2	3
STA OP,X	95	ZPX	2	4
STA OP	8D	ABS	3	4
STA OP,X	9D	ABX	3	5
STA OP,Y	99	ABY	3	5
STA (OP,X)	81	(ZP,X)	2	6
STA (OP),Y	91	(ZP),Y	2	6

■ STX : store X-register
schreibe X-Registerinhalt in Speicher

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
STX OP	86	ZP	2	3
STX OP,Y	96	ZPY	2	4
STX OP	8E	ABS	3	4

■ STY : store Y-register
schreibe Y-Registerinhalt in Speicher

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
STY OP	84	ZP	2	3
STY OP,X	94	ZPX	2	4
STY OP	8C	ABS	3	4

■ TAX : transfer akku to X-register
schreibe Akkuinhalt ins X-Register

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
TAX	AA	–	1	2

```
■ TAY : transfer akku to Y-register
schreibe Akkuinhalt ins Y-Register

Flags: N Z C I D V
       + +

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:   Takte:

TAY          A8          -            1       2
-----------------------------------------------------------
■ TSX : transfer stackregister to X-register
schreibe Stackregisterinhalt ins X-Register

Flags: N Z C I D V
       + +

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:   Takte:

TSX          BA          -            1       2
-----------------------------------------------------------
■ TXA : transfer X-register to akku

Flags: N Z C I D V
       + +

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:   Takte:

TXA          8A          -            1       2
-----------------------------------------------------------
■ TXS : transfer X-register to stackregister
schreibe X-Registerinhalt ins Stackregister

Flags: N Z C I D V
       keine

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:   Takte:

TXS          9A          -            1       2
-----------------------------------------------------------
■ TYA : transfer Y-register to akku
schreibe Y-Registerinhalt in Akku

Flags: N Z C I D V
       + +

Addressierungsarten:
Assembler:   Hex-Code:   Abkürzung:   Byte:   Takte:

TYA          98          -            1       2
-----------------------------------------------------------
```

Tabelle 2. Die Befehle des 6510-Prozessors (Schluß)

```
-----------------------------------------------------------
■ ABS : absolute (absolut)
Der Operand ist eine vierstellige, hexadezimale
Adresse.

Beispiel:
LDA $C000
Der Inhalt der Adresse $C000 wird in den Akku geladen.

-----------------------------------------------------------
■ ABX : absolut X-indiziert
Der Operand ist eine vierstellige hexadezimale Zahl.
Der Inhalt des X-Registers wird zum Operanden addiert
und ergibt die Arbeits-Adresse.

Beispiel:
LDX #$10
LDA $C000,X
Der Inhalt der Speicherstelle $C010 ($C000 + $0010)
wird in den Akku geladen.

-----------------------------------------------------------
■ ABY : absolut Y-indiziert
Der Operand ist eine vierstellige hexadezimale Zahl.
Der Inhalt des Y-Registers wird zum Operanden addiert
und ergibt die Arbeits-Adresse.

Beispiel:
LDY #$10
LDA $C000,Y
Der Inhalt der Speicherstelle $C010 ($C000 + $0010)
wird in den Akku geladen.

-----------------------------------------------------------
```

```
>>Byte<< : In den Tabellen 2 und 3 gibt diese Spalte
die jeweilige Länge des kompletten Befehls mit
Operand an.

-----------------------------------------------------------
>>Flags<< : einzelne Bits des Statusregisters

N : negative flag. Zeigt an, daß bei einer Operation
einer der beiden Operanden zwischen $80 (128) und $FF
(255) liegt, also das letzte Bit gesetzt ist.

Z : zero flag. Zeigt an, daß das Ergebnis einer
Operation im Akku gleich Null ist.

C : carry flag. Zeigt an, daß bei einer Operation ein
Übertrag entstanden ist.

I : interrupt flag. Durch Setzen dieses Bit lassen sich
Interrupts unterbinden.

D : decimal flag. Durch Setzen dieses Bit wir der
Prozessor in den Dezimalmodus geschaltet. Das bedeutet,
daß zum Beispiel das Ergebnis der Addition von $09 und
$01 nicht $0A, sondern $10 ergibt.

V : overflow flag (Überlauf). Zeigt an, daß das
Ergebnis einer Operation größer $FF (=255) war.

-----------------------------------------------------------
■ IM : immediate (unmittelbar)
Die Adressierungsart >>immediate<< bedeutet, daß der
Operand unmittelbar als Wert weiterverarbeitet wird.

Beispiel:
LDA #$00
Die hexadezimale Zahl $00 wird direkt in den Akku
geladen.

-----------------------------------------------------------
■ OP : Operand
Je nach Adressierungsart besteht der Operand eines
Befehls aus einem (Adressierung >>immediate<< und
>>zeropage<<) oder zwei Byte (>>absolute<<).

-----------------------------------------------------------
■ ZP : zeropage
Der Operand besteht aus einem Byte und gibt eine
Adresse in der Zeropage (Speicherbereich $0000 bis
$00FF) an.

Beispiel:
LDA $2B
Der Inhalt der Speicherstelle $002B wird in den Akku
geladen.

-----------------------------------------------------------
■ ZPX : Zeropage X-indiziert
Der Inhalt des X-Registers wird zum zweistelligen,
hexadezimalen Operanden addiert. Das Ergebnis ist eine
Adresse in der Zeropage (Speicherbereich $0000 bis
$00FF).

Beispiel:
LDX #$05
LDA $43,X
Der Inhalt der Adresse $0048 ($0043 + $0005) wird in
den Akku geladen.

-----------------------------------------------------------
■ ZPY : Zeropage Y-indiziert
Der Inhalt des Y-Registers wird zum zweistelligen,
hexadezimalen Operanden addiert. Das Ergebnis ist eine
Adresse in der Zeropage (Speicherbereich $0000 bis
$00FF).

Beispiel:
LDY #$05
LDA $43,Y
Der Inhalt der Adresse $0048 ($0043 + $0005) wird in
den Akku geladen.

-----------------------------------------------------------
■ (ZP,X) : indiziert indirekt
Der Inhalt des X-Registers wird zum zweistelligen,
hexadezimalen Operanden addiert und ergibt eine Adresse
in der Zeropage (Speicherbereich $0000 bis $00FF).
Deren Inhalt und der Inhalt der darauffolgenden Adresse
ergibt in der Form Lo-Byte/Hi-Byte die Arbeitsadresse.

Beispiel:
Adresse $20 hat den Inhalt $00
Adresse $21 hat den Inhalt $C0
LDX #$0E
LDA ($12,X)
```

Tabelle 3. Diese Abkürzungen werden in den Tabellen 1 und 2 verwendet

```
Der Inhalt der Zeropage-Adressen $0020 ($000E + $0012)
und $0021 ergibt die Arbeits-Adresse $C000. Deren
Inhalt wird in den Akku geladen.
-------------------------------------------------------
■ (ZP),Y : indirekt indiziert
Der zweistellige, hexdezimale Operand ergibt eine
Adresse in der Zeropage (Speicherbereich $0000 bis
$00FF). Deren Inhalt und der Inhalt der darauffolgenden
Speicherstelle ergibt in der Form Lo-Byte/Hi-Byte eine
Adresse, zu der der Inhalt des Y-Registers addiert
wird. Das Ergebnis ist die Arbeitsadresse.
```

```
Beispiel:
Adresse $20 hat den Inhalt $00
Adresse $21 hat den Inhalt $C0
LDY #$10
LDA ($20),Y
Der Inhalt der Adresse $C010 ($C000 + $0010) wird in
den Akku geladen.
-------------------------------------------------------
```

Tabelle 3. Diese Abkürzungen werden in den Tabellen 1 und 2 verwendet (Schluß)

ROM-Routinen in eigenen Programmen

Das Rad ist schon erfunden! Ähnlich verhält es sich mit verschiedenen Routinen, die ein Assembler-Programmierer immer wieder benötigt. Aber warum soll man sich die Arbeit des Programmierens machen, wenn das Betriebssystem viele ständig benötigte Routinen schon enthält und man nur noch zu wissen braucht, ab welcher Adresse sie stehen?

Angenommen, Sie möchten in Assembler einige komplexe Dinge programmieren wie beispielsweise eine neue mathematische Funktion (wie wäre es mit dem Kotangens) und diese auf dem Bildschirm ausgeben. Das ist eine große Aufgabe, zu der zunächst einmal die Übernahme des Arguments in das Maschinenprogramm, dann einige Fließkomma-Rechenoperationen und schließlich die Ausgabe auf dem Bildschirm geschrieben werden müßten, wenn da nicht schon fast alles an verborgener Stelle als fertige Programm-Module im Computer vorhanden wäre!

Sowohl im unteren (von $A000 bis $BFFF) als auch im oberen ROM-Bereich (von $E000 bis $FFFF) liegt die Firmware (Software, die hardwaremäßig integriert ist) fest verschachtelt vor. Der untere ROM-Abschnitt wird Basic-Interpreter, der obere ROM-Bereich Betriebssystem genannt, wobei diese Einteilung aber den Kern der Sache nicht genau trifft, denn Interpreter, Editor und Betriebssystem führen ein gemischtes Dasein, quer durch alle genannten ROM-Bereiche.

Mindestens fünf Informationen braucht ein Assembler-Programmierer, wenn er das breite Programmangebot des ROMs nutzen möchte:

1. Einsprungadresse
2. Format der Eingabeparameter
3. Adressen der Eingabeparameter
4. Adressen der Ausgabeparameter
5. Format der Ausgabeparameter

Nicht alle Routinen, die man benutzen kann, erfordern alle fünf Informationen, manche weniger, einige auch mehr, und schließlich gibt es noch Programmroutinen, die den Aufruf einer oder sogar mehrerer anderer Routinen notwendig machen.

In der Tabelle 1 sind – nach Anwendungen sortiert – die wichtigsten Firmware-Möglichkeiten mit den erforderlichen Ein- und Ausgabeparametern aufgeführt. Das sind natürlich beileibe nicht alle. Die Auswahl erfolgte subjektiv! Es sind einfach diejenigen, die mir bislang am häufigsten untergekommen sind. Außerdem wurde auf die Kernel-Routinen verzichtet: Man findet diese sehr gut dokumentiert bereits in einer Reihe von Büchern und im Kurs »Von Basic zu Assembler« in diesem Sonderheft.

Die Tabelle nennt den Label-Namen, die Einsprungadresse und gibt eine Kurzbeschreibung der Funktion. Das Ein- und das Ausgabeformat ist ebenso angegeben wie die Adressen, an denen diese Parameter übergeben werden. Die verwendeten Bezeichnungen halten sich eng an die im Assembler-Kurs kennengelernten. Sie sind allgemein üblich:

FAC	Fließkomma-Akku 1
ARG	Fließkomma-Akku 2
A	Akkumulator
X,Y	X-, Y-Register
A/Y	2-Byte-Angabe im Format LSB/MSB im Akku/Y-Register
FLPT	Fließkommazahl im Normalformat
MFLPT	gepacktes Fließkommaformat

Damit das alles nicht so trocken abläuft, soll noch ein kleines Beispiel vorgestellt werden! Die oben erwähnte Kotangens-Funktion wird in einem Maschinenprogramm erzeugt, das durch USR anzuspringen ist. In Bild 1 finden Sie ein Flußdiagramm zu dem Programm, welches hier als Hypra-Ass-Listing abgebildet ist (Listing 1). Ein kurzes Testprogramm liefert Listing 2.

Der Einsprung mittels USR bietet den Vorteil, daß der Übergabewert gleich im FLPT-Format im FAC »landet«. Es ist aber sinnvoll, den Übergabeparameter mittels der MOVMF-Routine zu »retten«, weil durch die Kosinus-Funktion der FAC verändert wird. Wenn auch das Ergebnis der Kosinus-Funktion mittels MOVMF beiseitegelegt wurde, holen wir durch MOVFM den Anfangswert wieder in den FAC und bilden mittels SIN den Sinus davon. Schließlich teilen wir den im Speicher stehenden Kosinuswert durch den im FAC befindlichen Sinuswert (unter Verwendung von FDIV). Das Ergebnis ist der Kotangens:

COT X = (COS X/SIN X)

Dieser Wert befindet sich nun im FAC und wird mit dem RTS an das Basic-Programm zurückgeliefert. Im Testprogramm weisen wir ihm dann die Variable E zu.

Dieses kurze Beispiel soll Ihnen den Mund wäßrig machen. Sehr viel detaillierter werden die ROM-Routinen im Kurs »Von Basic zu Assembler« in diesem Sonderheft behandelt werden.

(Heino Ponnath/sk)

Literatur:

1. Kassera/Kassera, Programmieren in Maschinensprache, München 1985: Markt&Technik Verlag, MT 830
2. West, C 64 Computerhandbuch, München 1984, Te-wi
3. Babel/Krause/Dripke, Das Interface Age Systemhandbuch zum C 64, München 1983: Interface Age Verlag
4. Ponnath, C 64 Wunderland der Grafik, München 1985: Markt&Technik Verlag MT 756.

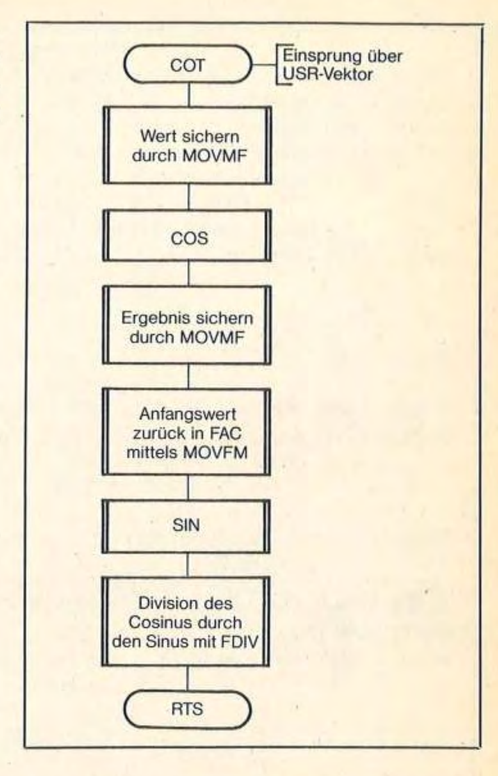
```
hypra-ass  assemblerlisting:

                10   -          .li 1,4,7
                20   -          .ba $6000
;
;einsprung mittels usr
;zuvor usr-vektor einstellen!
;
                160  -          .eq cos=$e264
                165  -          .eq movfm=$bba2
                170  -          .eq movmf=$bbd4
                180  -          .eq sin=$e26b
                190  -          .eq fdiv=$bb0f
                200  -          .eq wert=$7000
                205  -          .eq wert1=$7010
;
6000 a210    :212  -start     ldx #<(wer;t1)
6002 a070    :214  -          ldy #>(wert1)
6004 20d4bb  :216  -          jsr movmf
6007 2064e2  :220  -          jsr cos
600a a200    :230  -          ldx #<(wert)
600c a070    :240  -          ldy #>(wert)
600e 20d4bb  :250  -          jsr movmf
6011 a910    :252  -          lda #<(wert1)
6013 a070    :254  -          ldy #>(wert1)
6015 20a2bb  :256  -          jsr movfm
6018 206be2  :260  -          jsr sin
601b a900    :270  -          lda #<(wert)
601d a070    :280  -          ldy #>(wert)
601f 200fbb  :290  -          jsr fdiv
6022 60      :300  -          rts
;
                320  -          .sy 1,4,7

symbols in alphabetical order:

cos           = $e264
fdiv          = $bb0f
movfm         = $bba2
movmf         = $bbd4
sin           = $e26b
start         = $6000
wert          = $7000
wert1         = $7010

end of assembly  0:25.9
base = $6000  last byte at $6022
```

Listing 1. Hypra-Ass-Listing der Kotangens-Funktion

```
10 REM***TEST FUER COTANGENS***
20 POKE785,0:POKE786,96:REM USR-VEKTOR
30 INPUT"WINKEL";W:W=W*π/180:REM AUF BOGENMASS
40 E=USR(W):REM AUFRUF DES PROGRAMMES
50 PRINTW,E:REM ERGEBNIS IN E
60 END
READY.
```

Listing 2. Test der Kotangens-Funktion

Bild 1. Flußdiagramm einer Kotangens-Funktion



1. Routinen, die die Kooperation von Basic und Assembler erleichtern:

Label	Adresse	Funktion	Eingabe Format	Eingabe Adresse	Ausgabe Format	Ausgabe Adresse
CHRGET	0073	Holt nächstes Byte	1 Byte	Basic-Text	1 Byte	A
CHRGOT	0073	Holt aktuelles Byte	1 Byte	Basic-Text	1 Byte	A
READY	A474	Erzeugt READY-Status	-	-	-	-
LINGET	A96B	Holt Integerwert (0-63999)	ASCII-Zahl	Basic-Text	2-Byte Integer	14/15
FRMNUM	AD8A	Holt beliebigen numerischen Ausdruck	Basic-Ausdruck	Basic-Text	FLPT	FAC
FRMEVL	AD9E	Holt beliebigen Ausdruck	Basic-Ausdruck	Basic-Text	a) bei Fließkomma: FLPT	FAC
					b) bei Integer: FLPT	FAC
					c) bei String: Zeiger auf Descriptor	FAC+3, FAC+4

Diese Routine setzt außerdem eine Reihe von Flaggen:
VALTYP($0D) 0=Zahl FF=String
INTFLAG ($0E) 0=Fließkomma 80=integer

		War Ausdruck einfache Variable, dann zeigt VARNAM ($45/6) das 1. Byte des Variablen-Namens				
CHKCLS	AEF7	Prüft auf ») «	ASCII	Basic-Text	-	-
CHKOPN	AEFA	Prüft auf » («	ASCII	Basic-Text	-	-
CHKCOM	AEFD	Prüft auf »,«	ASCII	Basic-Text	-	-
SYNCHR	AEFF	Prüft auf Zeichen	ASCII	Basic-Text	-	-
		im Akkumulator		A		
		Diese 4 Routinen überlesen das Zeichen, wenn vorhanden. Wenn nicht vorhanden, folgt SYNTAX ERROR				
ISVAR	AF28	Sucht Variablenwert	Name +	$45/46	a) Zahl:	
			Kennung		FLPT	FAC
					b) String:	
					Descriptor-FAC+3	
ORDVAR	B0E7	Sucht Variablennamen	Name+	$45/46		
			Kennung		Adresse $47/48	
GTBYTC	B79B	Holt Zahl (0-255)	ASCII	Basic-Text	1 Byte	X
GETNUM	B7EB	Liest 2 Integerzahlen	ASCII	Basic-Text	2Byte-Int.	$14/15
		(Trennung durch Komma)			1Byte-Int.	X
		1. Zahl: 0 bis 65535				
		2. Zahl: 0 bis 255				
COMBYT	E200	Prüft auf »,« und holt	ASCII	Basic-Text	1 Byte	X
		folgende Zahl				

2. Routinen, die Verschiebungen im Speicher durchführen:

BLTUC	A3BF	Verschiebt Blöcke	Adressen:			
			Quelle			
			Start	$5F/60		
			Ende+1	$5A/5B		
			Ziel			
			Ende+1	$58/59	-	-
PUTINT	A9C4	Schiebt FAC als Integer	FLPT	FAC	2Byte-	angegebene
		in Variable	Adresse	$49/50	Integer	Variable
PTFLPT	A9D6	Schiebt FAC	FLPT	FAC	MFLPT	angegebene
		in Variable	Adresse	$49/50		Variable
GETSPT	AA2C	Schiebt String-	Zeiger	FAC+3		
		descriptor in Variable	Adresse	$49/50	Descriptor	angegebene Variable
STRVAL	B7B5	Zahlenstring in	ASCII	ab $22	FLPT	FAC
		FAC einlesen	Länge	A		
CONUPK	BA8C	Lädt ARG aus Speicher	MFLPT	A/Y	FLPT	ARG
MOVFM	BBA2	Lädt FAC aus Speicher	MFLPT	A/Y	FLPT	ARG
MOVMF	BBD4	Schiebt FAC	FLPT			
		in Speicher	Adresse	FAC X/Y	MFLPT	ange-gebener Speicher
MOVFA	BBFC	ARG in FAC kopieren	FLPT	ARG	FLPT	FAC
MOVAF	BC0C	FAC in ARG kopieren	FLPT	FAC	FLPT	ARG
ACTOFC	BC3C	Akku in FAC schieben	1Byte	A	FLPT	FAC

3. Routinen zur Arithmetik:

ASCADD	AA27	Addiert ASCII-Ziffer	ASCII	A	FLPT	FAC
		zu FAC	Ziffer			
OROP	AFE6	FAC=(FAC)OR(ARG)	FLPT	FAC,ARG	FLPT	FAC
ANDOP	AFE9	FAC=(FAC)AND(ARG)	FLPT	FAC,ARG	FLPT	FAC
			0	Y		
FACINX	B1AA	FAC wird als Integer	FLPT	FAC	2Byte-	A/Y
		in A/Y abgelegt			Integer	
UMULT	B357	16-Bit-Multiplikation	2-Byte-Integer			
			Zahl1	$28/29	2Byte-	X/Y
			Zahl2	$71/72	Integer	
CIVAYF	B391	Integer (-32768 bis	2Byte-	A/Y	FLPT	FAC
		32767) in FAC	Integer			
SGNFT	B3A2	Integer (0 bis 255)	1Byte	y	FLPT	FAC
		in FAC				
GETADR	B7F7	Wandelt FAC zu	FLPT	FAC	2Byte-	Y/A
		Integer (0-65535)			Integer	+$14/15
FADDH	B849	FAC = FAC + 0,5	FLPT	FAC	FLPT	FAC
FSUB	B850	FAC=Speicherzahl	MFLPT	Zeiger A/Y	FLPT	FAC
		-FAC				
			FLPT	FAC		
FSUBT	B853	FAC = ARG - FAC	FLPT	ARG,FAC	FLPT	FAC
FADD	B867	FAC=Speicherzahl	MFLPT	Zeiger A/Y	FLPT	FAC
		+FAC	MFLPT	FAC		
FADDT	B86A	FAC = ARG + FAC	FLPT	ARG,FAC	FLPT	FAC
COMPLT	B947	Erzeugt Zweier-	FLPT	FAC	FLPT	FAC
		komplement von FAC				
LOG	B9EA	FAC = ln(FAC)	FLPT	FAC	FLPT	FAC
FMULT	BA28	FAC=Speicherwert•FAC	MFLPT	Zeiger A/Y	FLPT	FAC
			FLPT	FAC		
FMULTT	BA30	FAC = ARG • FAC	FLPT	ARG,FAC	FLPT	FAC
MUL10	BAE2	FAC = 10 • FAC	FLPT	FAC	FLPT	FAC
DIV10	BAFE	FAC = FAC/10	FLPT	FAC	FLPT	FAC
FDIVF	BB07	FAC=ARG/Speicher-	MFLPT	Zeiger A/Y	FLPT	FAC
		zahl	FLPT	ARG		
FDIV	BB0F	FAC=Speicherzahl/FAC	MFLPT	Zeiger A/Y	FLPT	FAC
			FLPT	FAC		
FDIVT	BB14	FAC = ARG/FAC	FLPT	FAC,ARG	FLPT	FAC
SIGN	BC28	Ermittelt Vorzeichen	FLPT	FAC	1Byte	A
		von FAC			1 → +	
					0 → 0	
					FF → -	
ABS	BC58	FAC = ABS(FAC)	FLPT	FAC	FLPT	FAC
FCOMP	BC5B	Vergleicht FAC mit	MFLPT	Zeiger A/Y	1Byte:	A
		Speicherzahl	FLPT	FAC	1: FAC > Speicher	
					0: FAC = Speicher	
					FF: FAC < Speicher	
INT	BCCC	FAC = INT(FAC)	FLPT	FAC	FLPT	FAC
AADD	BD7E	Addiert A zu FAC	FLPT	FAC	FLPT	FAC
			1Byte	A		
SQR	BF71	FAC = SQR(FAC)	FLPT	FAC	FLPT	FAC
MPOT	BF78	FAC=Speicherwert	FLPT	FAC	FLPT	FAC
		↑ FAC				
			MFLPT	Zeiger A/Y		
FPWRT	BF7B	FAC = ARG ↑ FAC	FLPT	ARG,FAC	FLPT	FAC
NEGOP	BFB4	FAC = -FAC	FLPT	FAC	FLPT	FAC
EXP	BFED	FAC = e↑FAC	FLPT	FAC	FLPT	FAC
POLYX	E059	Polynomberechnung	Adresse	Zeiger A/Y	FLPT	FAC
		$FAC=a0+a1x+a2x^2+...$ Zeiger weist auf Start der Konstantentabelle. 1. Byte = Polynomgrad. Weitere Bytes sind die Koeffizienten des Polynoms in der Reihenfolge an,......,a0 im MFLPT-Format.				
COS	E264	FAC = COS(FAC)	FLPT	FAC	FLPT	FAC
SIN	E26B	FAC = SIN(FAC)	FLPT	FAC	FLPT	FAC
TAN	E2B4	FAC = TAN(FAC)	FLPT	FAC	FLPT	FAC
ATN	E30E	FAC = ATN(FAC)	FLPT	FAC	FLPT	FAC

4. Auswahl von Ein-/Ausgabe-Routinen:

ERROR	A437	Fehlermeldung aus-	Fehler-	X	ASCII	Bildschirm
		geben und READY	nummer			
LIST	A69C	Listet Basic-Programm	-	-	-	-
NUMDON	AABC	Druckt FAC auf	FLPT	FAC	ASCII	Bildschirm
		Bildschirm aus				
STROUT	AB1E	Gibt String auf Bild-	Adresse	Zeiger A/Y	ASCII	Bildschirm
		schirm aus. Ende=0				
SYNERR	AF08	Ausgabe SYNTAX ERROR	-	-	ASCII	Bildschirm
OVERR	B97E	Ausgabe OVERFLOW ERR.	-	-	ASCII	Bildschirm
LINPRT	BDCD	Druckt Integerzahl	2Byte-	X/A	ASCII	Bildschirm
		(0 bis 65535) aus.	Integer			
FACOUT	BDD7	Druckt FAC auf	FLPT	FAC	ASCII	Bildschirm
		Bildschirm aus				
FOUT	BDDD	FAC wird zu ASCII-	FLPT	FAC	ASCII	ab $100
		String (Ende=0).				(Ende=0)
		Kann direkt mit STROUT				Startadr.
		ausgegeben werden.				A/Y
SAVET	E156	Save	Parameter aus Basic-Text			
VERFYT	E165	Verify	Parameter aus Basic-Text			
LOADT	E168	Load	Parameter aus Basic-Text			
SLPARA	E1D4	Holt Parameter für Save, Verify, Load aus dem Basic-Text				
PLOTK	E50A	Setzt Cursorposition	Zeile	X		
			Spalte	Y		
HOME	E566	Cursor in Home-Position				
PLOTR	E56C	Setzt Cursor-Position	Zeile	$D6		
			Spalte	$D3		
GETKBC	E5B4	Holt Zeichen aus	-	-	1Byte	A
		Tastaturpuffer				
PRT	E716	Gibt Zeichen in A	1Byte	A	ASCII	Bildschirm
		auf Bildschirm aus				
CLRLN	E9FF	Löscht xte Bild-	Zeilen-	X	-	-
		schirmzeile	nummer			

Tabelle der ROM-Routinen (Schluß)

Checksummer V3 und MSE

Diese beiden Programme sind unentbehrlich beim Abtippen unserer Listings. Sie helfen, Tippfehler vor allem bei Maschinenprogrammen zu vermeiden und sparen eine Menge Zeit.

Nobody is perfect. Jeder Computer-Fan, egal ob blutiger Anfänger oder ausgefuchster Profi, macht beim Abtippen von Programmen Tippfehler. Diese Fehler später zu finden, kann ein langwieriges Unterfangen sein.

Deshalb haben wir für Sie die Programme »Checksummer V3« und »MSE« (MaschinenSpracheEditor) entwickelt. Der Checksummer ist für Basic-Programme und der MSE für Maschinensprache-Listings zuständig.

Der Checksummer

Zuerst einmal müssen Sie das Checksummer-Programm (siehe Listing 1) abtippen. Dabei sollten Sie äußerst sorgfältig vorgehen, vor allem bei den Zahlen in den DATA-Zeilen 20 bis 30. Wenn Sie trotzdem noch einen Tippfehler gemacht haben, meldet sich das Programm später mit einem entsprechenden Hinweis. Wenn Sie fertig sind, speichern Sie das Programm auf Diskette oder Kassette.

Jetzt geht es los:

1. Starten Sie den Checksummer durch die Eingabe von »RUN« und das Drücken der RETURN-Taste.
2. Wenn die Meldung »Checksummer aktiviert...« auf dem Bildschirm erscheint, haben Sie keinen Tippfehler gemacht und der Checksummer ist nun eingeschaltet.
3. Zum Löschen des Basic-Programms geben Sie bitte »NEW« ein. Keine Angst, der Checksummer selbst wird dadurch nicht gelöscht.
4. Nun können wir den Checksummer testen. Geben Sie bitte folgende Zeile ein und drücken Sie die RETURN-Taste:

1 REM

In der linken oberen Bildschirmecke sehen Sie nun die Prüfsumme über die eben eingegebene Basic-Zeile. Sie muß <63> lauten. Dem Checksummer ist es übrigens egal, ob Sie »1 REM« oder »1REM« eintippen. Nur innerhalb von Anführungszeichen ist die richtige Anzahl an Leerzeichen wichtig. Diese Prüfsummen erscheinen (sofern Sie den Checksummer eingeschaltet haben) immer dann, wenn Sie eine Basic-Zeile eintippen und dann die RETURN-Taste drücken. In der 64'er finden Sie die Prüfsumme immer am Ende jeder Programmzeile.

```
10 PRINT"CHECKSUMMER FUER C 64"
11 PRINT:PRINT"EINEN MOMENT, BITTE ..."
12 FOR I=828 TO 864:READ A:POKE I,A:PS=PS+
   A:NEXT I
13 IF PS<>5765 THEN PRINT"TIPPFEHLER IN DE
   N ZEILEN 20 BIS 22":END
14 SYS 828:PS=0:FOR I=58464 TO 58583:READ
   A:POKE I,A:PS=PS+A:NEXT I
15 IF PS<>16147 THEN PRINT"TIPPFEHLER IN D
   EN ZEILEN 22 BIS 30":END
16 POKE 1,53:POKE 42289,96:POKE 42290,228
17 PRINT"CHECKSUMMER AKTIVIERT."
18 PRINT:PRINT" AUSSCHALTEN : POKE1,55 ODE
   R"SPC(27)"<RUN/STOP+RESTORE>"
19 PRINT:PRINT" ANSCHALTEN : POKE1,53"
20 DATA 169,0,133,254,162,1,189,93,3,133,2
   55,160,0,177,254
21 DATA 145,254,136,208,249,230,255,165,25
   5,221,95,3,208,238,202
22 DATA 16,230,96,160,224,192,0,160,2,169,
   0,170,133,254,177
23 DATA 95,240,40,201,32,208,3,200,208,245
   ,133,255,138,41,7
24 DATA 170,240,14,72,165,255,24,42,105,0,
   202,208,249,133,255
25 DATA 104,170,232,165,255,24,101,254,133
   ,254,76,111,228,192,4
26 DATA 48,219,198,214,165,214,72,162,3,16
   9,32,157,1,4,189
27 DATA 212,228,32,210,255,208,12,0,92,72,
   32,201,255,170,104
28 DATA 144,1,138,96,202,16,228,166,254,16
   9,0,32,205,189,169
29 DATA 62,32,210,255,104,133,214,32,108,2
   29,169,141,32,210,255
30 DATA 76,128,164,9,60,18,19
© 64'er
```

Listing 1. Der »Checksummer 64 V3« für Basic-Listings

```
5 PRINT CHR$(14)                                   <242>
10 PRINT"{CLR}"                                     <254>
20 PRINT"Q******************************W"          <130>
30 PRINT"{4DOWN,2SPACE}TEST{SPACE,BLUE,6SP
   ACE}"                                            <022>
40 PRINT"BBBBBBBBBBBBBBBBBBBBBBBBBBBB"              <108>
© 64'er
```

Bild 1. Die Bedeutung der Steuerzeichen wird im nachfolgenden Text erklärt

In Zeile 10 müssen Sie nach den Anführungszeichen die Tasten <SHIFT CLR/HOME> drücken und nicht die Klammern mit dem Wort CLR eingeben. In Zeile 20 drücken Sie nach den Anführungszeichen die CBM-Taste und den Buchstaben <Q>, gefolgt von mehreren SHIFT- und Stern-Tasten und zum Schluß die CBM-Taste und den Buchstaben <W>. In Zeile 30 ist es viermal die CURSOR-abwärts-Taste, gefolgt von zweimaliger Leertaste, dann <SHIFT T> und normal EST, zum Schluß noch einmal die Leertaste, die Farbtaste Blau <CTRL 7> und sechsmal die Leertaste. Zeile 40 besteht lediglich aus mehreren Grafikzeichen, die mit der CBM-Taste und <B> erzeugt werden.

CTRL steht für Control-Taste, so bedeutet {CTRL+A}, daß Sie die Control-Taste und die Taste »A« drücken müssen. Im folgenden steht:

{DOWN}	Taste neben rechtem Shift, Cursor unten	{SPACE}	Leertaste	{RVSON}	Control-Taste & 9
{UP}	Shift-Taste & Taste neben rechtem Shift; Cursor hoch	{SHIFT-Space}	Shift-Taste & Leertaste	{RVOFF}	Control-Taste & 0
		{F1} bis {F8}	Funktionstasten	{ORANGE}	Commodore-Taste & 1
		{RETURN}	Return-Taste	{BROWN}	Commodore-Taste & 2
{CLR}	Shift-Taste & 2. Taste ganz rechts oben	{BLACK}	Control-Taste & 1	{LIG.RED}	Commodore-Taste & 3
		{WHITE}	Control-Taste & 2	{GREY 1}	Commodore-Taste & 4
{INST}	Shift-Taste & Taste ganz rechts oben	{RED}	Control-Taste & 3	{GREY 2}	Commodore-Taste & 5
		{CYAN}	Control-Taste & 4	{LIG.GREEN}	Commodore-Taste & 6
{HOME}	2. Taste von ganz rechts oben	{PURPLE}	Control-Taste & 5	{LIG.BLUE}	Commodore-Taste & 7
{DEL}	Taste ganz rechts oben	{GREEN}	Control-Taste & 6	{GREY 3}	Commodore-Taste & 8
{RIGHT}	Taste ganz rechts unten	{BLUE}	Control-Taste & 7		
{LEFT}	Shift-Taste & Taste unten rechts	{YELLOW}	Control-Taste & 8		

Tabelle 1. Die Steuerbefehle in den Listings

Diese Zahlen dürfen Sie NICHT mit abtippen.
Als Beispiel sehen Sie Bild 1. Am rechten Rand jeder Spalte sehen Sie die Prüfsummen in eckigen Klammern.

Damit sind wir beim zweiten wichtigen Punkt: Sehen Sie sich die Zeile 240 von Listing 2 genauer an. Nach dem ersten Anführungszeichen nach dem PRINT-Befehl sehen Sie eine geschweifte Klammer { }. Immer, wenn Sie in einem unserer Listings diese Klammern sehen, dürfen Sie das, was innerhalb der Klammern steht, nicht eintippen. Sie müssen die entsprechende Taste drücken. Beispiel:
10 PRINT "{CLR}"
bedeutet: Nach dem Anführungszeichen die »Bildschirmlöschen«-Taste drücken (<SHIFT CLR/HOME>). In Tabelle 1 sehen Sie eine Zusammenfassung aller möglichen Steuertasten mit dem entsprechenden Klartext.

Weiterhin sehen Sie in Bild 1 (Bedeutung der Steuerzeichen) in Zeile 30 ein unterstrichenes »T« nach der Klammer. Das bedeutet, daß Sie ein »T« zusammen mit der SHIFT-Taste drücken müssen, also <SHIFT T>. Wenn ein Zeichen »überstrichen« ist, müssen Sie dieses zusammen mit der CBM-Taste eingeben. Die CBM-Taste befindet sich ganz links unten auf der Tastatur und hat die Aufschrift »C=«.

```
100 REM DIESES PROGRAMM ERZEUGT DEN          <210>
110 REM MSE V1.1 AUF DISKETTE.               <039>
120 REM BESITZER EINER DATASETTE             <178>
130 REM MUESSEN DIE '8' AM ENDE VON          <145>
140 REM ZEILE 343 IN EINE '1' AENDERN!       <176>
150 REM                                      <212>
230 IF PEEK(44)<>32 THEN PRINT"{CLR}SIE HA
    BEN VERGESSEN, DIE POKES EINZUGE- BEN!
    ":END                                    <050>
240 PRINT"{CLR}";:DIM H(75):FOR I=Ø TO 9     <042>
250 H(48+I)=I:H(65+I)=I+1Ø:NEXT:Z=1ØØØ       <136>
260 FOR I=2048 TO 3755 STEP 2Ø:PRINT"{HOME
    }ICH LESE ZEILE:"Z                       <253>
261 FOR N=Ø TO 19:READ A$:IF LEN(A$)<>2 TH
    EN 9ØØ                                   <062>
262 IF PEEK(63)+PEEK(64)*256<>Z THEN 8ØØ     <011>
270 H=ASC(LEFT$(A$,1)):L=ASC(RIGHT$(A$,1))   <199>
280 D=H(H)*16+H(L):S=S+D:POKE I+N,D          <165>
290 NEXT:READ V:IF S<>V THEN 9ØØ             <139>
300 S=Ø:Z=Z+1:NEXT:R=PEEK(2111):H=PEEK(21Ø
    6)                                       <126>
301 POKE 5328Ø,R:POKE 53281,H:POKE 646,R:P
    RINT"{CLR}DIE DATA-ZEILEN SIND FEHLERF
    REI!"                                    <080>
302 PRINT"SIE KOENNEN NUN DIE FARBEN DES M
    SE"                                      <209>
303 PRINT"EINSTELLEN.":PRINT"{2DOWN,SPACE,
    RVSON}DRUECKEN SIE <1>, <2> ODER <9>     <205>
304 PRINT"{DOWN,2SPACE}<1> - RAHMEN-/SCHRI
    FTFARBE                                  <013>
305 PRINT"{2SPACE}<2> - HINTERGRUNDFARBE     <233>
306 PRINT"{DOWN,2SPACE}<9> - FARBEN UEBERN
    EHMEN                                    <158>
307 PRINT"{2DOWN}FARBE <1>:"R:PRINT"FARBE
    <2>:"H                                   <066>
308 GET A:IF A=Ø THEN 3Ø8                    <210>
309 IF A=1 THEN R=(R+1)AND 15                <098>
310 IF A=2 THEN H=(H+1)AND 15                <086>
311 IF A=9 THEN 34Ø                          <217>
312 GOTO 3Ø1                                 <034>
340 POKE 21Ø6,H:POKE 2111,R                  <153>
342 POKE 631,19:POKE 632,13:POKE 198,2       <135>
343 PRINT"{CLR}SAVE"CHR$(34)"MSE V1.1"CHR$
    (34)",8                                  <091>
344 POKE 43,1:POKE 44,8:POKE 45,172:POKE 4
    6,14:END                                 <140>
800 PRINT"{CLR,RVSON}SIE HABEN ZEILE"Z"{LE
    FT,SPACE}VERGESSEN:":A=PEEK(646)AND 15   <124>
810 POKE 646,PEEK(53281)AND 15:PRINT"LIST"
    Z-2"-"Z+2:POKE 646,A                     <224>
820 GOTO 92Ø                                 <082>
900 PRINT"{CLR,RVSON}SIE HABEN EINEN TIPPF
    EHLER GEMACHT:":A=PEEK(646)AND 15        <154>
910 POKE 646,PEEK(53281)AND 15:PRINT"LIST"
    Z:POKE 646,A                             <173>
920 POKE 631,19:POKE 632,17:POKE 633,13:PO
    KE 198,3:END                             <126>
1000 DATA ØØ,ØB,Ø8,ØA,ØØ,9E,32,3Ø,36,31,ØØ
     ,ØØ,ØØ,A2,Ø8,A9,36,85,A4,A9, 1247       <119>
1001 DATA Ø8,85,A5,A9,ØØ,85,A6,A9,BØ,85,A7
     ,AØ,ØØ,B1,A4,91,A6,C8,DØ,F9, 2888       <054>
1002 DATA E6,A5,E6,A7,CA,DØ,F2,A9,36,85,Ø1
     ,4C,ØØ,BØ,2Ø,D1,B1,A9,ØØ,8D, 2781       <096>
1003 DATA 21,DØ,A9,ØF,8D,2Ø,DØ,8D,86,Ø2,AØ
     ,B3,A9,74,2Ø,FF,B1,AØ,B3,A9, 2679       <089>
1004 DATA B9,2Ø,FF,B1,AØ,ØØ,2Ø,CF,FF,99,Ø1
     ,Ø2,C8,C9,ØD,DØ,F5,88,FØ,D2, 2912       <217>
1005 DATA CØ,11,9Ø,Ø2,AØ,1Ø,8C,ØØ,Ø2,2Ø,EA
     ,B1,AØ,B3,A9,CF,2Ø,FF,B1,2Ø, 2327       <045>
1006 DATA 8E,B4,85,FC,85,62,2Ø,8E,B4,85,FB
     ,85,61,2Ø,A7,B4,DØ,2Ø,AØ,B3, 2864       <199>
1007 DATA A9,E5,2Ø,FF,B1,2Ø,8E,B4,85,6Ø,2Ø
     ,8E,B4,85,5F,2Ø,A7,B4,DØ,ØA, 2624       <091>
1008 DATA A5,61,C5,5F,A5,62,E5,6Ø,9Ø,Ø6,2Ø
     ,43,B3,4C,3A,BØ,A9,AA,AØ,ØØ, 2379       <167>
1009 DATA EA,EA,E6,FB,DØ,Ø2,E6,FC,2Ø,3F,B2
     ,9Ø,EF,4C,FB,B4,A2,Ø2,86,58, 319Ø       <041>
1010 DATA A9,A6,AØ,9D,2Ø,F2,B1,2Ø,E4,FF,FØ
     ,FB,C9,3Ø,9Ø,ØC,C9,47,BØ,Ø8, 297Ø       <231>
1011 DATA C9,3A,9Ø,ØB,C9,41,BØ,Ø7,C9,14,DØ
     ,ØF,4C,ØB,B1,2Ø,D2,FF,A6,58, 2322       <121>
1012 DATA 95,F7,C6,58,DØ,D2,6Ø,AE,8D,Ø2,FØ
     ,26,C9,ØC,DØ,Ø3,4C,ØB,B6,C9, 2685       <057>
1013 DATA 13,DØ,Ø3,4C,8B,B5,C9,ØD,DØ,Ø3,4C
     ,BA,B4,C9,1Ø,DØ,Ø3,4C,68,B5, 2282       <225>
1014 DATA C9,ØE,DØ,Ø6,2Ø,5F,B4,4C,64,B1,4C
     ,92,BØ,A5,F9,2Ø,Ø2,B1,ØA,ØA, 2132       <208>
1015 DATA ØA,ØA,85,F9,A5,F8,2Ø,Ø2,B1,Ø5,F9
     ,6Ø,C9,3A,9Ø,Ø2,69,Ø8,29,ØF, 195Ø       <092>
1016 DATA 6Ø,A6,59,EØ,Ø8,9Ø,1F,A6,58,EØ,Ø2
     ,BØ,Ø6,2Ø,D2,FF,4C,8E,BØ,C6, 25Ø9       <188>
1017 DATA 59,AØ,14,A9,92,2Ø,F2,B1,CA,DØ,FA
     ,84,57,68,68,4C,8B,B1,A6,D3, 2891       <197>
1018 DATA EØ,Ø8,BØ,Ø3,4C,92,BØ,2Ø,D2,FF,A6
     ,58,EØ,Ø2,9Ø,Ø9,C6,59,2Ø,D2, 2468       <049>
1019 DATA FF,C6,58,DØ,F9,4C,8E,BØ,48,4A,4A
     ,4A,4A,2Ø,59,B1,68,29,ØF,C9, 2419       <035>
1020 DATA ØA,9Ø,Ø2,69,Ø6,69,3Ø,4C,D2,FF,A2
     ,FC,9A,2Ø,D1,B1,2Ø,48,B2,2Ø, 2261       <073>
1021 DATA EA,B1,2Ø,9F,B2,A5,FC,2Ø,4E,B1,A5
     ,FB,2Ø,4E,B1,2Ø,ED,B1,A9,3A, 286Ø       <148>
1022 DATA AØ,2Ø,2Ø,F2,B1,A9,ØØ,85,59,2Ø,8E
     ,BØ,2Ø,ED,B1,A4,59,2Ø,EF,BØ, 253Ø       <233>
1023 DATA 91,FB,C8,84,59,CØ,Ø8,9Ø,EC,2Ø,1Ø
     ,B2,A9,12,2Ø,D2,FF,2Ø,8E,BØ, 2657       <105>
1024 DATA 2Ø,EF,BØ,C5,FF,FØ,ØD,2Ø,43,B3,A9
     ,14,AØ,14,2Ø,F2,B1,4C,A2,B1, 2665       <034>
1025 DATA A9,92,2Ø,D2,FF,2Ø,33,B2,2Ø,EØ,B2
     ,2Ø,3F,B2,9Ø,9F,4C,8B,B5,A9, 2648       <123>
1026 DATA 93,2Ø,D2,FF,A2,ØØ,A9,Ø3,9D,ØØ,D8
     ,9D,ØØ,D9,9D,ØØ,DA,9D,ØØ,DB, 2476       <237>
1027 DATA E8,DØ,EF,6Ø,A9,ØD,2C,A9,2Ø,4C,D2
     ,FF,2Ø,D2,FF,98,4C,D2,FF,2Ø, 2965       <160>
1028 DATA E4,FF,FØ,FB,6Ø,84,5D,85,5C,AØ,ØØ
     ,B1,5C,FØ,Ø6,2Ø,D2,FF,C8,DØ, 31ØØ       <077>
1029 DATA F6,6Ø,A5,FB,85,5A,AØ,ØØ,84,5B,B1
     ,FB,18,65,5A,85,5A,9Ø,Ø2,E6, 26Ø6       <156>
1030 DATA 5B,Ø6,5A,26,5B,C8,CØ,Ø8,9Ø,EC,A5
     ,5A,65,5B,85,FF,6Ø,18,A5,FB, 2467       <219>
1031 DATA 69,Ø8,85,FB,9Ø,Ø2,E6,FC,6Ø,A5,FB
     ,C5,5F,A5,FC,E5,6Ø,6Ø,AØ,B3, 31Ø6       <183>
1032 DATA A9,FB,2Ø,FF,B1,AØ,Ø1,B9,ØØ,Ø2,2Ø
     ,D2,FF,CC,ØØ,Ø2,C8,9Ø,F4,A9, 2692       <098>
1033 DATA 14,ED,ØØ,Ø2,AA,2Ø,ED,B1,CA,DØ,FA
     ,A5,62,2Ø,4E,B1,A5,61,2Ø,4E, 2457       <060>
1034 DATA B1,2Ø,ED,B1,A5,6Ø,2Ø,4E,B1,A5,5F
     ,2Ø,4E,B1,EA,EA,EA,EA,EA,EA, 3122       <190>
1035 DATA EA,EA,24,5E,1Ø,Ø1,6Ø,A9,12,2Ø,D2
     ,FF,A2,28,2Ø,ED,B1,CA,DØ,FA, 27Ø3       <087>
1036 DATA A9,92,4C,D2,FF,A5,D6,C9,16,BØ,Ø1
     ,6Ø,A9,AØ,85,A4,A9,78,85,A6, 2945       <204>
1037 DATA A9,Ø4,85,A5,85,A7,A2,13,AØ,27,B1
     ,A4,91,A6,88,1Ø,F9,CA,FØ,19, 2671       <208>
1038 DATA 18,A5,A4,69,28,85,A4,9Ø,Ø2,E6,A5
     ,18,A5,A6,69,28,85,A6,9Ø,EØ, 25Ø3       <251>
1039 DATA E6,A7,4C,B6,B2,A9,91,4C,D2,FF,A9
     ,ØF,8D,18,D4,A9,ØØ,8D,Ø5,D4, 2776       <000>
1040 DATA A9,F7,8D,Ø6,D4,A9,11,8D,Ø4,D4,A9
     ,32,8D,Ø1,D4,A9,ØØ,8D,ØØ,D4, 2413       <126>
1041 DATA AØ,8Ø,2Ø,Ø9,B3,A9,1Ø,8D,Ø4,D4,6Ø
     ,A2,FF,CA,DØ,FD,88,DØ,F8,6Ø, 2914       <240>
1042 DATA A9,ØF,8D,18,D4,A9,2D,8D,Ø5,D4,A9
     ,A5,8D,Ø6,D4,A9,21,8D,Ø4,D4, 2385       <119>
1043 DATA A9,Ø7,8D,Ø1,D4,A9,Ø5,8D,ØØ,D4,AØ
```

```
     ,FF,2Ø,Ø9,B3,A9,2Ø,8D,Ø4,D4, 225Ø    <Ø78>
1Ø44 DATA A9,ØØ,8D,Ø1,D4,8D,ØØ,D4,6Ø,38,2Ø
     ,FØ,FF,8A,48,98,48,18,AØ,Ø6, 2179    <175>
1Ø45 DATA A2,18,2Ø,FØ,FF,AØ,B4,A9,ØA,2Ø,FF
     ,B1,2Ø,12,B3,2Ø,E4,FF,FØ,FB, 2931    <Ø93>
1Ø46 DATA A2,1D,A9,14,2Ø,D2,FF,CA,DØ,FA,68
     ,A8,68,AA,18,4C,FØ,FF,ØD,ØD, 27Ø4    <Ø88>
1Ø47 DATA ØD,2Ø,2Ø,2Ø,2Ø,2Ø,2Ø,2Ø,4D,41,53
     ,43,48,49,4E,45,4E,53,5Ø,52, 1144    <216>
1Ø48 DATA 41,43,48,45,2Ø,2D,2Ø,45,44,49,54
     ,4F,52,2Ø,ØD,ØD,2Ø,2Ø,2Ø,2Ø, 1Ø23    <Ø38>
1Ø49 DATA 2Ø,2Ø,2Ø,2Ø,56,4F,4E,2Ø,4E,2E,4D
     ,41,4E,4E,2Ø,26,2Ø,44,2E,57, 1128    <2Ø6>
1Ø5Ø DATA 45,49,4E,45,43,4B,ØØ,ØD,ØD,ØD,2Ø
     ,2Ø,2Ø,5Ø,52,4F,47,52,41,4D, 11Ø2    <117>
1Ø51 DATA 4D,4E,41,4D,45,2Ø,3A,2Ø,ØØ,ØD,ØD
     ,2Ø,2Ø,2Ø,53,54,41,52,54,41, 1Ø73    <Ø95>
1Ø52 DATA 44,52,45,53,53,45,2Ø,3A,2Ø,24,ØØ
     ,ØD,ØD,2Ø,2Ø,2Ø,45,4E,44,41, 1Ø14    <129>
1Ø53 DATA 44,52,45,53,53,45,2Ø,2Ø,2Ø,3A,2Ø
     ,24,ØØ,92,Ø1,Ø1,5Ø,52,4F,47, 1136    <228>
1Ø54 DATA 52,41,4D,4D,2Ø,3A,2Ø,ØØ,12,2Ø,2Ø
     ,2A,2A,2A,2Ø,46,41,4C,53,43, 1Ø24    <Ø27>
1Ø55 DATA 48,45,2Ø,45,49,4E,47,41,42,45,2Ø
     ,2A,2A,2A,2Ø,2Ø,92,ØØ,ØD,ØD, 1Ø58    <Ø98>
1Ø56 DATA 2A,2A,2A,2Ø,45,4E,44,45,2Ø,2A,2A
     ,2A,ØØ,13,Ø1,2Ø,2Ø,12,44,92, 916     <153>
1Ø57 DATA 49,53,4B,2Ø,4F,44,45,52,2Ø,12,54
     ,92,41,5Ø,45,ØD,ØØ,13,2Ø,2Ø, 1151    <Ø35>
1Ø58 DATA 49,2F,4F,2Ø,2D,2Ø,46,45,48,4C,45
     ,52,ØØ,2Ø,D1,B1,2Ø,48,B2,AØ, 16Ø6    <Ø12>
1Ø59 DATA B3,A9,CF,2Ø,FF,B1,2Ø,8E,B4,85,FC
     ,2Ø,8E,B4,85,FB,C5,61,A5,FC, 32Ø7    <251>
1Ø6Ø DATA E5,62,9Ø,23,A5,FB,C5,5F,A5,FC,E5
     ,6Ø,BØ,19,2Ø,A7,B4,DØ,14,6Ø, 286Ø    <112>
1Ø61 DATA 2Ø,A7,B4,FØ,ØC,85,F9,2Ø,A7,B4,FØ
     ,Ø5,85,F8,4C,EF,BØ,68,68,2Ø, 2749    <Ø88>
1Ø62 DATA 43,B3,4C,5F,B4,2Ø,CF,FF,C9,4C,DØ
     ,Ø9,2Ø,D1,B1,2Ø,48,B2,4C,ØB, 2372    <Ø46>
1Ø63 DATA B6,C9,ØD,6Ø,A9,ØØ,85,5E,2Ø,5F,B4
     ,2Ø,EA,B1,2Ø,ØD,B5,24,5E,3Ø, 2Ø42    <12Ø>
1Ø64 DATA Ø5,2Ø,E4,FF,FØ,FB,2Ø,E1,FF,FØ,26
     ,2Ø,9F,B2,24,5E,1Ø,Ø9,2Ø,4E, 2435    <198>
1Ø65 DATA B5,2Ø,ØD,B5,2Ø,6Ø,B5,2Ø,33,B2,2Ø
     ,3F,B2,9Ø,D7,AØ,B4,A9,28,2Ø, 219Ø    <2Ø7>
1Ø66 DATA FF,B1,2Ø,E4,FF,C9,ØD,DØ,F9,A9,ØØ
     ,85,5E,A5,61,85,FB,A5,62,85, 3Ø56    <24Ø>
1Ø67 DATA FC,2Ø,EØ,B2,4C,64,B1,A5,FC,2Ø,4E
     ,B1,A5,FB,85,FF,2Ø,4E,B1,A9, 3ØØ3    <221>
1Ø68 DATA 2Ø,AØ,3A,2Ø,F2,B1,AØ,ØØ,2Ø,ED,B1
     ,B1,FB,2Ø,4E,B1,C8,CØ,Ø8,9Ø, 2566    <Ø7Ø>
1Ø69 DATA F3,2Ø,ED,B1,24,5E,3Ø,Ø3,A9,12,2C
     ,A9,2Ø,2Ø,D2,FF,2Ø,1Ø,B2,A5, 219Ø    <Ø59>
1Ø7Ø DATA FF,2Ø,4E,B1,A9,92,2Ø,D2,FF,4C,EA
     ,B1,A9,FF,85,B8,85,B9,A9,Ø4, 3Ø73    <Ø29>
1Ø71 DATA 85,BA,2Ø,CØ,FF,A2,FF,4C,C9,FF,2Ø
     ,CC,FF,A9,FF,4C,C3,FF,2Ø,5F, 3315    <189>
1Ø72 DATA B4,A9,8Ø,85,5E,2Ø,4E,B5,2Ø,48,B2
     ,A2,24,A9,2D,2Ø,D2,FF,CA,DØ, 2596    <111>
1Ø73 DATA FA,2Ø,EA,B1,2Ø,EA,B1,2Ø,6Ø,B5,4C
     ,C1,B4,2Ø,B8,B5,A6,5F,A4,6Ø, 2812    <Ø15>
1Ø74 DATA A9,61,2Ø,D8,FF,BØ,ØA,2Ø,B7,FF,29
     ,BF,DØ,Ø3,4C,FB,B4,A9,Ø1,2Ø, 2577    <2Ø1>
1Ø75 DATA C3,FF,2Ø,68,B6,AØ,B4,A9,4F,2Ø,FF
     ,B1,2Ø,F9,B1,4C,FB,B4,2Ø,68, 2921    <237>
1Ø76 DATA B6,A9,37,AØ,B4,2Ø,FF,B1,2Ø,F9,B1
     ,A2,Ø8,C9,44,FØ,Ø6,A2,Ø1,C9, 2717    <213>
1Ø77 DATA 54,DØ,F1,A9,Ø1,A8,2Ø,BA,FF,AØ,ØØ
     ,EØ,Ø1,FØ,1A,A9,4Ø,8D,2Ø,Ø2, 24Ø3    <1Ø1>
1Ø78 DATA A9,3A,8D,21,Ø2,B9,Ø1,Ø2,99,22,Ø2
     ,C8,CC,ØØ,Ø2,9Ø,F4,C8,C8,DØ, 2182    <127>
1Ø79 DATA ØC,B9,Ø1,Ø2,99,2Ø,Ø2,C8,CC,ØØ,Ø2
     ,DØ,F4,98,A2,2Ø,AØ,Ø2,4C,BD, 2Ø18    <Ø25>
1Ø8Ø DATA FF,2Ø,B8,B5,A5,BA,C9,Ø8,9Ø,33,A6
     ,B9,86,57,A9,Ø1,2Ø,C3,FF,A9, 28ØØ    <Ø22>
1Ø81 DATA 6Ø,85,B9,2Ø,CØ,FF,BØ,28,A5,BA,2Ø
     ,B4,FF,A5,B9,2Ø,96,FF,2Ø,A5, 2911    <Ø53>
1Ø82 DATA FF,85,61,A5,9Ø,4A,4A,BØ,13,2Ø,A5
     ,FF,85,62,2Ø,AB,FF,A5,57,85, 2663    <214>
1Ø83 DATA B9,A9,ØØ,2Ø,D5,FF,9Ø,Ø3,4C,A3,B5
     ,86,5F,84,6Ø,A5,BA,C9,Ø1,DØ, 2639    <131>
1Ø84 DATA ØA,AD,3D,Ø3,85,61,AD,3E,Ø3,85,62
     ,4C,FB,B4,A9,13,2Ø,D2,FF,A2, 23ØØ    <12Ø>
1Ø85 DATA 1C,2Ø,ED,B1,CA,DØ,FA,6Ø,ØØ,ØØ,ØØ
     ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ, 123Ø    <143>
```


Listing 2. Der MSE-Lader

Der MSE

Der MSE dient zur Eingabe von Maschinensprache-Programmen. Als erstes müssen Sie den sogenannten »MSE-Lader« (Listing 2) abtippen. Dieser erzeugt erst das eigentliche MSE-Programm auf Diskette oder Kassette.

Wichtig: Vor dem Eintippen des MSE-Laders müssen Sie unbedingt ein paar Befehle eingeben (ohne Basic-Zeilennummer): POKE 44,32 : POKE 8192,0 : NEW

Jetzt können Sie beginnen, das Listing 2 abzutippen. Der MSE-Lader erkennt zwar, wenn Sie beim Eintippen der DATA-Zeilen einen Fehler gemacht haben, aber wenn Sie ganz sicher gehen möchten, sollten Sie den Checksummer vor dem Eintippen aktivieren. Die Prüfsummen für den MSE-Lader finden Sie am Ende der jeweiligen Programmzeilen.

Wenn Sie das Listing 2 nicht auf einmal abtippen möchten, müssen Sie vor jedem neuen Laden des Programms unbedingt die oben genannte POKE-Zeile eingeben!

Wenn Sie alles richtig gemacht haben und das Programm fehlerfrei abgetippt wurde, speichert es sich nach dem Starten selbst auf Diskette oder Kassette unter dem Namen »MSE V1.0«. Dieses fertige MSE-Programm laden Sie dann bei Bedarf wie ein normales Basic-Programm und starten es mit »RUN«.

So arbeitet man mit dem MSE

Als erstes möchte der MSE den Namen des zu bearbeitenden Programms wissen. Dieser steht in der ersten Zeile unserer MSE-Listings. Dann müssen Sie die Start- und Endadresse des Programms eingeben. Dies sind die letzten beiden, vierstelligen Hexadezimalzahlen in der ersten Zeile unserer Listings.

Wenn Sie ein Programm von Diskette oder Kassette laden wollen, um an einer bestimmten Stelle weiterzutippen oder noch eine Korrektur vorzunehmen, geben Sie auf die Frage nach der Startadresse ein »L« ein. Danach müssen Sie <D> oder <T> drücken, je nachdem, ob Sie von Diskette oder Kassette (»tape«) laden möchten. Wenn das Programm unter diesem Namen nicht auf der Diskette vorhanden ist oder ein sonstiger Ladefehler vorlag, meldet sich der MSE mit »I/O-ERROR«. In diesem Fall drücken Sie <RUN/STOP RESTORE> und geben einfach noch einmal »RUN« ein.

Beim Abtippen geben Sie nach und nach die abgedruckten Buchstaben und Zahlen des jeweiligen Listings ohne die Freiräume dazwischen ein. Wenn Sie in einer Zeile einen Tippfehler gemacht haben, meldet sich der MSE sofort mit einem Brummton und der Meldung »EINGABE-FEHLER«. Nach einem Druck auf die RETURN-Taste können Sie mit der DEL-Taste den Fehler korrigieren. Wenn Sie das gewünschte Programm vollständig eingegeben haben, speichert es der MSE automatisch auf Diskette oder Kassette.

Bei längeren Listings ist es unwahrscheinlich, daß Sie das komplette Programm auf einmal eingeben. Sie können Ihre bisherige Tipparbeit jederzeit durch <CTRL S> auf Diskette oder Kassette speichern und Ihr Werk später fortsetzen. Sie sollten sich dann allerdings im Heft markieren, wie weit Sie beim Abtippen gekommen sind! Später geben Sie dann nach dem Laden des ersten Programmteils <CTRL N> ein und auf die dann folgende Frage nach der Startadresse die Zeilennummer (Adresse), bei der Sie aufgehört haben zu tippen.

<CTRL M> erlaubt Ihnen jederzeit, Ihr Werk listen zu lassen. Durch <SPACE> können Sie weiterlisten lassen und durch <RUN/STOP> das Listen abbrechen.

Wenn Sie einen Drucker besitzen, können Sie das Programm auch mit <CTRL P> ausdrucken. Mit <CTRL L> wird das Programm noch einmal neu in Ihren C 64 geladen.

(F. Lonczewski/N. Mann/D. Weineck/tr)

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Geschäftsführender Chefredakteur: Michael Scharfenberger

Chefredakteur: Albert Absmeier
Stellv. Chefredakteur: Georg Klinge
Redaktion: Gottfried Knechtel (kn), Klaus Schrödl (sk)

Layout: Leo Eder (Leitung), Rolf Raß (Cheflayouter)
Andrea Miller, Katja Milles

Fotografie: Jens Jancke

Titelgestaltung: Andrea Miller

Produktionsleiter: Klaus Buck

Anzeigenverkaufsleitung: Ralph-Peter Rauchfuss

Anzeigenverkauf: Britta Fiebig (282)

Auslandsrepräsentation:

Schweiz: Markt&Technik Vertriebs AG,
Kollerstr. 3, CH-6300 Zug,
Tel. 042-41 56 56, Telex: 862329

USA: M&T Publishing Inc.; 501 Galveston Drive Redwood City,
CA 94063
Telefon: (415) 366-3600

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten werden, so muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt&Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt&Technik Verlag AG verlegten Publikationen und dazu, daß Markt&Technik Verlag AG Geräte und Bauteile nach der Bauanleitung herstellen läßt und vertreibt oder durch Dritte vertreiben läßt. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Marketingleiter: Hans Hörl (114)

Vertriebsleiter: Helmut Grünfeldt (189)

Anzeigenverwaltung und Disposition: Lisa Landthaler (233)

Druck: SOV St. Otto-Verlag GmbH,
Laubanger 23, 8600 Bamberg

Bezugsmöglichkeiten: Leser-Service: Telefon (089) 4613-249. Bestellungen nimmt der Verlag oder jede Buchhandlung entgegen.

Preis: Das Einzelheft kostet DM 14,–

Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: Pegasus Buch- und Zeitschriften-Vertriebs GmbH, Hauptstätter Straße 96, 7000 Stuttgart 1, Telefon (0711) 6483-0

Verantwortlich:
Für redaktionellen Teil: Albert Absmeier
Für Anzeigen: Britta Fiebig

Redaktionsdirektor: Michael M. Pauly

Vorstand: Carl-Franz von Quadt, Otmar Weber

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen:
Markt&Technik Verlag Aktiengesellschaft,
Hans-Pinsel-Straße 2, 8013 Haar bei München,
Telefon (089) 4613-0, Telex 5-22052

ISSN 0931-8933